Main Page

Welcome to CS3773: Java as a Second Language.

This is the main page for CS3773, Spring 2008, taught by Simson Garfinkel at the Naval Postgraduate School.

We are currently on Week 11: The Big Finish

From here you can explore the following links:

- Syllabus
- Week-by-week Outline
- Assignments
- Things to do (if you want to hack the wiki)

Contents

News

March 11

- Final projects are now in the Final Projects Catagory

March 10

- Interesting in figuring out wall intersections in Flatland?
- Please feel free to enter which Section of the Bloch Book you wish to discuss.

March 9

- Answers to Quiz 9 have been uploaded.

- Quiz 9 stats: Minimum score: 40 ; Average: 58 ; Max Score: 80

March 7

- You should get assignment 8 in before the end of the quarter; shoot for Monday March 10th.

March 3

- Details regarding the Final Project have been posted
- Week7 can now be submitted; it will display as an applet on the validate page. (Code is experimental)
- Link to the short "Warriors of the Web".
- An Office Hours Sign-Up page has been created.

March 1st

Quiz 8 grades have been posted. Class mean was 88; stddev was 25.81

February 29

Please enter your requests for topics over the next two weeks in the Coming Attractions section.

February 25

- Prof. Herzog provided these slides from his lecture on Friday.
- Slides from Spring 99 have been uploaded to this server

February 22

- Final Project Ideas have been posted.

February 21

- Friday's class will be taught by Professor Jonathan Herzog. He will be discussing Othello, Timers, and XMLRPC.
- Next week we will be building a basic chat system using the XMLRPC system. Ambitious students may also build an Othello game.
- Please submit your 1-paragraph proposal for the final project by Friday. The ideas will be summarized and placed on a web page (without your name) for other students to see.
- Homework #7, due February 24th, is now optional. You may submit your Homework #6 working in a web page.
- Homework #8, due March 3, will be the Desktop Search application.
- Homework #9, due March 10th, will be the Chat/Othello program. This is your last homework.
- Final Projects will be due on March 27th.

February 12

- Apparently DoD policy allows session-tracking cookies, just not persistent cookies.

February 10

- You are reminded to submit your homework through the class submission website.

- Be sure that there are no Subversion Conflicts in the code that you submit.
- Your homework should make it clear what you did that is special. One way to do this is by including a README file.

February 6

- A Layout Demo has been added to the week5 subversion repository.
- I found an interesting article about Java Anonymous Classes at Developer.com.

February 4

- Homework #3 and #4 grades have gone out.
- If you are not submitting homework, you are missing out on free credit. Try to solve every problem.
- Media:Week5.pdf Week 5 slides are posted.

January 31

- Homework #4 can now be submitted using the new submission system
- Making the jarfile in the "hw4" package:

Because homework #4 is in the 'hw4' package, you must do the following:

1. Put the line "package hw4;" at the beginning of each of your files.
2. Put your files in a directory called "hw4"
3. add the files to the jarfile with this command:

```
jar cfv yourname.jar hw4/*.java hw4/*.class
```

January 26

- Equations of Motion have been posted to help you with the electrostatics.
- If you are willing to have your homework used as a class example on debugging on Monday, please drop me an email. Let me know if I can use your name or if you want to show your homework anonymously. Slgarfin 09:46, 27 January 2008 (PST)
- If you are having Null reference exceptions, you may find this discussion of null reference exceptions useful.
- If you haven't done it lately, be sure to update your netbeans 6.0 and get the new fixes.

January 23

- week3 source code has been posted. You can check it out using this command line:

```
svn checkout http://domex.nps.edu/cs3773/svn/ cs3773
```

January 18

- There was a bug in the homework submission validation function, but it's fixed now.

Old News

- The Homework Submission Link now works.
- Need a laptop for this class? Please send your requests to Professor Mantak Shing.
- Week 2 slides (first draft) Are uploaded.
- Week 2 thursday slides are also uploaded.
- Please take a look at the Talk:Blog
- A Bug in Week 02 code has been discovered. Click on the wiki page for the fix.

Main Page

Contributed Resources

- Useful Websites

Please feel free to contribute to this wiki, either by editing pages (as appropriate) or contributing to the web-based discussion. You can log into this wiki using your NPS username and password.

CS 3773

- Main Page

- Assignments
- Readings

Wiki

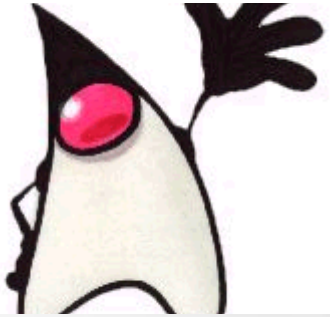
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Syllabus

Lectures will be held at Every day at 0800 in GE-117.

Although portions of this class may be taught using tele-presence equipment, this class will not be videotaped. The making of video or audio recordings is strictly prohibited.

Contents

Contact Information

Course Location	GE 117
Meeting Time:	MTWR 0800-0850 F 0800-0950
Professor:	Simson L. Garfinkel, Ph. D.
Phone:	831-656-7602 (office); 617-876-6111 (home)
NPS e-mail:	slgarfin
Office Hours:	Tuesdays, 0900-1000
Professor's Website:	http://www.simson.net

Catalog Description

CS3773 Java as a Second Language (4-2)

A first course in Java for students experienced in another programming language. Students learn to implement problem solutions using the procedural and object-oriented language features of Java. Topics include: program structures and environment, arrays, exceptions, constructors and finalizers, class extension, visibility and casting, overriding versus overloading, abstract classes and interfaces, files and streams, class loaders, threads, and sockets. Programming projects provide students the opportunity to implement techniques covered in class. Prerequisite: Recent completion of the complete series in another

programming language course, or programming experience in another programming language.

Learning Outcomes

Upon successful completion of this course, you will be able to:

- Use NetBeans, Sun's integrated development environment (IDE) for Java.
- Decompose a problem using classes and objects.
- Identify, understand and use all of the Java reserved keywords.
- Use Java Container Classes and Iterators, including Array, Vector, Stack, Hashtable, and BitSet.
- Develop classes and methods that match a specified signature.
- Create a JAR file that contains all information necessary to run a Java program.
- Write programs that use files and streams.
- Interact with an SQL database using JDBC.
- Write a program that displays a graphical user interface using Swing.
- Write an applet that displays itself in a web page
- Write an XMLRPC client and server.
- Develop a program that uses Lucene, the open source indexing kit.

Course Format

This course is divided into 11 weeks. Each week will follow roughly this schedule:

- Monday 0800 - The week's assignment is released. Class commences at 0800.
- Monday & Tuesday - New material is presented.
- Wednesday - Class discussion regarding the assigned reading.

- Thursday - Additional new material on the week's theme is presented.
- Friday 0800 - 0950 - Brief Quiz; Q&A; Lab time to work on the assignment.
- Sunday 1800 - Assignments are due at 1800, submitted electronically. Late assignments are not accepted.

You should bring a laptop equipped with NetBeans and JDK 1.6 (Java 6) to each class and lab period.

Text

This course has three texts:

- *Sams Teach Yourself Java 6 in 21 Days (5th Edition)*, which can be ordered from Amazon. The book may also be available in the campus book store. This isn't the best book about Java, but it is serviceable and it covers Java 6.
- The Java Tutorial
- Papers that can be downloadable from this website

We also have some recommended readings:

- *Effective Java: Programming Language Guide* ,by Joshua Bloch. This book is highly recommended; it will teach you a lot about Java in particular and programming in general.
- *The Java ™ Programming Language, 4th Edition*, by James Gosling, Ken Arnold and David Holmes. This 928-page book is the definitive guide to Java™ 2 Standard Edition 5.0 (J2SE 5.0). Unfortunately it's very difficult to read and is really written for Java experts, not for Java beginners.

Assessment

Grades are calculated as follows:

Weekly assessment (11)	25%
Programming Assignments (11):	50%
Final Project:	25%
Class Participation:	priceless

All work in this class should conform to the CS3773 Java Style Guide.

Class Participation

This is an intensive workshop-style class. As such, class participation is an important part of the experience. You won't get a grade for your participation, but participating will help you to understand the material better.

Grades

Grades are based on an absolute scale:

A 90 to 100% of the total possible points

B 80 to 89% of the total possible points

C 70 to 79% of the total possible points

D 60 to 69% of the total possible points

F 0 to 59% of the total possible points

Collaboration, Plagiarism and Academic Integrity

It is strongly recommended that you discuss the readings and assignments with your classmates. You may wish to organize reading or study groups for this purpose. However, it is also expected that the homework you submit will be your own work. You may not collaborate on homework; collaboration on the final project is limited to approved groups.

Plagiarism in any form will not be tolerated in this course. This includes both direct plagiarism, in which you reprint code or words written by another person without reference, and to intellectual plagiarism, in which you present another person's ideas or argument as if they are your own.

Academic integrity on the part of U.S. and international officers and civilians participating in NPS programs is an important aspect of professional performance. For this reason, the provisions of NAVPGSCOLINST 5370.1C of the Academic Honor Code will be strictly enforced.

If you have questions about collaboration, plagiarism or academic integrity, please contact the class staff.

Citation Policy

It is possible that you may wish to reference articles, algorithms, or websites in the preparation of your assignments. Citations should include the author of the work being cited, its title, where it is located (usually a URL), the date it was written, and in the case of URLs, the date that you downloaded it. A URL without an author, title, publication title, and publication date is not an acceptable citation format.

Citations that are bare URLs will be ignored.

Protocols

Communication is a central part of every course. This section of the syllabus describes what we expect from your communications with your fellow students and the course staff.

Notifications

For announcements and assignments, the Web is our authoritative form of communication. Students are expected to check the home page for both news and assignments at least once a week. If you hear a rumor, check it there. If you miss an announcement, it should be on the home page.

Homework

All homework is due at the start of class on the day for which it is assigned. In most cases your homework will be uploaded to the course website. Late homework is not accepted except in extraordinary cases.

Course Outline

Here is the week-by-week outline:

- Week 01: Java Without Classes
- Week 02: Classes
- Week 03: Object-Oriented Design
- Week 04: I/O

- Week 05: Basic Graphics and GUI with AWT and Swing
- Week 06: More Swing
- Week 07: Unicode, Text & Internationalization
- Week 08: Threads and the System
- Week 09: Network programming with Java
- Week 10: Style and Performance
- Week 11: The Big Finish

You can get a more detailed outline on the Week-by-week Outline page.

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Week-by-week Outline

PRELIMINARY - - - SUBJECT TO CHANGE

You have 11 weeks to learn Java and object-oriented design.

Each week will have:

- Stuff to learn
- Programming homework, due Sunday night..
- Quiz, in Friday in Lab

Course Outline

Contents

Week 01: Java Without Classes

- Course Goals
- Understand Java and how it's different from C++
- Compile Hello World
- Turn in your first assignment.

Week 02: Classes

Language Goals

- What is a class? What is an instance? What is an object?
- Understanding inheritance: methods & variables
- Encapsulation: public variables vs. accessor methods
- Final: final methods & final variables
- mutable vs. immutable objects
- Enums

Environment Goals

- applets vs. stand-alone programs.
- GUI vs. command-line programs.
- What CLASSPATH does

- Using the debugger

Week 03: Object-Oriented Design

- Inheritance
- Interfaces vs. Abstract classes
- Nested Classes
- Enumerated Types
- overriden methods
- overloaded methods
- overriding vs. overloading
- packages
- Annotations

Week 04: I/O

- Exceptions - try/catch/finally/throws
- I/O Exceptions
- Keyboard input
- Streams, Buffered Streams
- Files
- Persistance
- SQL/MySQL/JDBC

Week 05: Basic Graphics and GUI with AWT and Swing

- Anonymous classes and pseudo-function pointers
- Swing
- HelloWorldSwing (anonymous classes)
- Basic GUI components
- Containment Hierarchy
- Frames, event-driven programming,
- Events & Listeners.
- http://java.sun.com/docs/books/tutorial/uiswing/components/jcomponent.html

Week 06: More Swing

- Applets
- Really understand Swing
- Building GUI's with the NetBeans GUI builder.

Week 07: Unicode, Text & Internationalization

- Understand ASCII, Latin1 and Unicode
- Learn about Lucene, the open source indexing kit.
- Build an application using Lucene that will index the files on your

computer.

Week 08: Threads and the System

- Threads
- Synchronization
- Serialization

Week 09: Network programming with Java

You should know this by the end of this week:

- Theory: TCP/IP, client/server computing, OSI Stack, Sockets
- SOAP, XMLRPC, and REST
- Downloading a web pages

You may wish to know this:

- JavaSocket, Server Socket

Week 10: Style and Performance

Week 11: The Big Finish

- Answer all outstanding questions
- Final Project Presentations
- Tuesday: SQL & JDBC
- Wednesday: JAVA Security

- Thursday: My favorite Tools

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

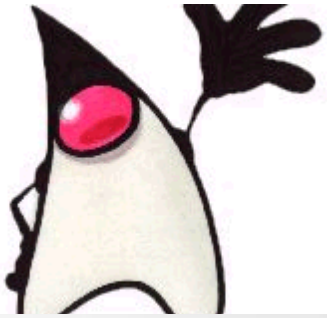
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Things to do

Review:

- http://en.wikipedia.org/wiki/Wikipedia:Text_editor_support
- http://qbnz.com/highlighter/index.php - GeSHI - Generic Syntax Highlighter

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

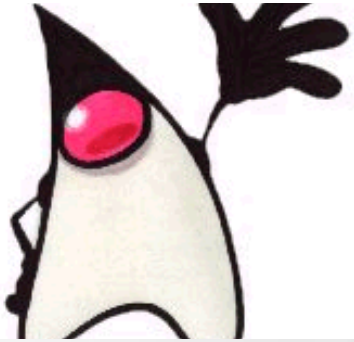
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Assignments

Assignment 1 - Hello World!

Assignment 2 - Building and validating a simple class '*Bot* ; Javadoc.

Assignment 3 - Abstract classes, packages, interfaces and more behaviors.

Assignment 4 - I/O and Exceptions. Some exercises involving exceptions, reading from the network, writing into an SQL database, and having fun with Java.

Assignment 5 - Flatland with a new GUI.

Assignment 6 - Reimplement the Flatland Control Panel with the GUI created with the Swing Builder..

Assignment 7 - **optional** Make your Flatland (or anything else) run as an applet..

Assignment 8 - Chat program with the Othello Server (due March 7th) .

Final Project - You create your own project. mini project. See the web page for ideas.

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

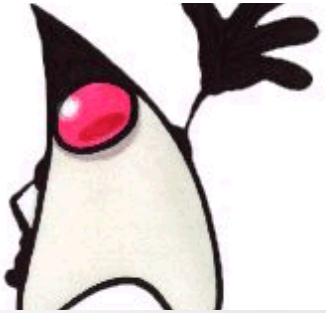
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Assignment 1

JavaTM Programming Tools and Documentation

Due date: Sunday, January 13, 6pm Pacific Time

Contents

Purpose

To set up the Java programming environment, learn the compilation process, and practice the submission process.

Grading

Assignment 1 is a chance to learn the grading process without having mistakes

counted against you. This assignment is graded pass/fail. If you turn it in, you get an "A". If you don't, you get an "F". It is suggested that you take this opportunity to familiarize yourself with the Java development environment, the assignment submission process, and the stringency of coding-style requirements.

One of the nice things about Java is that virtually all of the tools you need to develop Java programs available for free. What's more, there are many different free tools from different sources. Sun Microsystems invented Java and distributes a version called the Java Development Kit. The current JDK is number 6. The JDK includes a Java compiler (javac), the Java interpreter (java) a debugger (jdb), a documentation tool (javadoc) and many other tools. Unlike other development environments you may have used, these tools are designed to be used from the command line, like a DOS console, or UNIX xterm.

In addition to these command-line tools, there are a number of integrated development environments (IDEs) available for Java. These include NetBeans from Sun, Eclipse from the Eclipse Foundation, and EMACS, a text-mode editor distributed by the Free Software Foundation.

The purpose of this lab is to have you locate and install JDK 6 and other relevant tools on your computer.

Note:

- If you are using a Macintosh, JDK5 is pre-installed and JDK6 is not available.
- The PCs in the lab should have JDK 6 pre-installed.

Download the Software

1. Download and install JDK 6 Update 3 from Sun's website. (Note: if you have a Mac, then you will be using JDK 5, and it's pre-installed.)
2. Download and install NetBeans 6.0 from the NetBeans site.
3. The standard install places the JDK in the directory **c:\Program Files\Java \jdk1.6._03**. In order to use your newly installed JDK, you must ensure that your system is configured properly. There are three environment variable that must be properly setup in order to locate JDK components: **JAVA_HOME**, **PATH** and **CLASSPATH**.

Setting up Environment Variables on Windows

Once you have the JDK installed, you need to set up your computer's Environment Variables so that the JDK installation can be found. For reasons that defy comprehension these environment variables are not set up automatically but must be manually configured. On Windows this is done through the Environment Variables control panel which is reached by clicking the "Environment Variables" button on the Advanced tab of the System Properties panel. (On Mac/Unix/Linux this is done by modifying your .bash_profile, .cshrc, or .profile startup scripts.)

1. Right-click on the My Computer icon on your desktop and select **properties**
2. Click the Advanced Tab
3. Click the Environment Variables button
4. Assure that the following environment variables are set up:

JAVA_HOME *your JDK installation directory*

PATH *your JDK bin directory* needs to be in the PATH.

CLASSPATH	*You'll use this to tell your system where additional classes are installed.*

Information on setting JAVA_HOME and CLASSPATH on Windows 95/98

Subversion

Install Subversion on your computer and check out the repository from `http://domex.nps.edu/cs3773/svn/`.

Hello World

Using a text editor (like Notepad or EMACS), enter the following Java program.

Remember, Java is case sensitive so watch your capitalization.

```
/**
 * My first Java Program.
 *
 * @author Simson Garfinkel (change to your name)
 */

public class Assign1 {
   public static void main(String args[]) {
      System.out.println("Hello World!");
   }
}
```

Save the file as Assign1.java in the src directory. Open a command window. Change directories to the directory containing the source file you just created. Issue the following command, from the base <your login> directory, to compile your program:

```
javac Assign1.java
```

A successful compilation will create a Java class file named Assign1.class, consisting of Java byte codes. The "-d classes" directive tells the compiler to place compiled class files into the classes directory. This is the Java version of an object file created by other language compilers. You may execute this class file by invoking the Java interpreter (java) with the following command:

```
java Assign1
```

The Java interpreter searches the CLASSPATH list to find the Assign1.class file. It should find it in the classes directory. If you have any problems performing these two steps, you need to ensure that your PATH and CLASSPATH variable are set properly. In a DOS window you can see the values of all system variables by typing set.

Finally, we want to use the automatic documentation facility, javadoc. We want documentation of all files from the src directory to be in the docs directory, so run the command as follows:

```
javadoc –d docs *.java
```

All java commands can be run with –help, as javac –help to get a list of possible arguments.

Assign1 with the Add method

Some of the programs that you submit for this course will be graded with an automatic web-based grading system. Neat, huh? To get the sense of how it works, we want you to add a method called **calc** to your Assign1 class. The calc method should take two integer arguments and return a new integer.

Modify your class so it looks like this:

```
/**
 * My second Java Program.
 *
 * @author Simson Garfinkel (change to your name)
 */

public class Assign1 {
    public static int calc(int a,int b){
        return a+b;
    }
    public static void main(String args[]) {
        System.out.println("Hello World!");
        System.out.printf("3 + 4 = %d %n",Assign1.calc(3,4));
    }
}
```

The new **calc** method will add a+b and return the result. The new lines that we added to the **main()** static method will create an **Assign1** object and then run it's calc method with (3,4). The printf method looks a lot like the C printf method, with the exception that we use **%n** instead of **\n** as a newline.

Counter

Create a program called Counter which outputs the numbers 1 through 10. The program should be in a file called *Counter.java* and compiled into a file called *Counter.class*. Your output should look like this:

```
1
2
3
4
5
6
7
8
9
10
```

Primes

Create a program called Primes which accepts an argument on the command line and prints the prime numbers between 1 and that number. For example, if your program is called with the command:

```
java Primes 20
```

The output should be:

```
2 is prime
3 is prime
```

```
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
```

Upload

Bundle all of your files together into a JAR file with your username.

You can do this with the jar command, like this:

```
$ jar cvf slgarfin.jar *.java *.class
added manifest
adding: Assign1.java(in = 316) (out= 223)(deflated 29%)
adding: Counter.java(in = 268) (out= 199)(deflated 25%)
adding: Primes.java(in = 311) (out= 201)(deflated 35%)
adding: Assign1.class(in = 684) (out= 422)(deflated 38%)
adding: Counter.class(in = 406) (out= 295)(deflated 27%)
adding: Primes.class(in = 984) (out= 561)(deflated 42%)
$
```

Note: The dollar sign ($) is the prompt for the computer that I'm using.

Notice that we can run all of the programs out of the jar file by using the `-classpath` argument:

```
$ java -classpath slgarfin.jar Assign1
Hello World!
3 + 4 = 7
$ java -classpath slgarfin.jar Counter
```

```
1
2
3
4
5
6
7
8
9
10
$ java -classpath slgarfin.jar Primes 20
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
$
```

Here is what happens if you ask to run a function that isn't present:

```
$ java -classpath slgarfin.jar Count
Exception in thread "main" java.lang.NoClassDefFoundError: Count
```

Contest

For the contest, create Primes to print the number of primes between 1 and N, where N is an argument on the command line, if the "-total" argument is given. (This way you don't need to have two copies of Primes.java)

For example:

```
% java Primes -total 10
4

% java Primes -total 20
8
```

See Also

Assignments

Category: Assignments

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

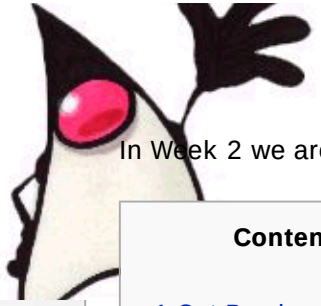
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



ent 2

In Week 2 we are learning all about Java classes.

Contents

Get Ready

Download the jarfile from https://domex.nps.edu/cs3773/week2/week2.jar

- Note: if you are using Internet Explorer, the week2.jar file may download as week2.zip; you will need to rename it.

You can unpack the file if you wish. It will have the following in it:

Flatland.java & Flatland.class

Source code and Bytecode for the Flatland class

FlatlandComponent.java & FlatlandComponent.class

Source code and byte code for the FlatlandComponent class (this is the GUI)

FlatlandDelegate.java & FlatlandDelegate.class

The interface that allows Flatland to communicate with the FlatlandComponent

FlatlandObject.java & FlatlandObject.class

Source code and byte code for the abstract superclass

Location.java & Location.class

Simple class that implements a place

Size.java & Size.class

Simple class that implements a size.

Build a Bot

In the assignment you will start with a class we have created called Bot. The Bot is a class for a simple robot. These area ideal robots which are 1 meter square. We'll be doing a lot with these little robots.

Our Bots live on a 2-dimensional plane called Flatland. The Flatland class keeps track of all of the objects that are in it, including the Bots, the Walls, and other objects that we haven't even thought about!

Your Bot class needs to have the following internal state:

- A location
- A size
- A speed
- A heading (0 = north, 90 = east, 180 = south, 270 = west)
- Color
- Speed

Your Bot class needs to implement the following methods:

- getCenter() (returns the Location of the center)
- getSize() (returns the size)

- setHeading(double heading)
- tick() (time passes)
- Bot(center,size) (constructor)
- toString() - turns it into a string

This specification can be packaged into either a FlatlandObject interface, which your Bot could implement, or a FlatlandObject abstract class, which your Bot would subclass. *What are the advantages or disadvantages of each?*

We have provided the javadoc output for the instructor's Bot class.

Test Your Bot

The Flatland class contains a **main()** function that you can use to test your Bot class. The simulation will create four bots with a starting location and move them around. If you unpack all of the files in the JAR file, you can run your simulation with this command:

```
java Flatland
```

Alternatively, you can run the simulation out of the jar file. Use this command:

```
java -classpath week2.jar:. Flatland
```

On Windows, I had to use this command:

```
java -classpath week2.jar;. Flatland
```

Note that you do not need to put your Bot.class file into the week2.jar file; you can have the Bot.class file in the current directory and the jar file specified by the classpath.

If you wish, you can visualize the simulation as well. This is done with the **FlatlandComponent** class. The class has a public static method called **launch**. Call it with the Flatland object as its argument and the simulation will start off; we won't discuss how the visualiation works for a few weeks.

This code automatically launches the visualization when the -gui flag is given to the Flatland class:

```
if(args.length==1 && args[0].equals("-gui")){
            FlatlandComponent.launch(fl);
        }
```

Get it?

Bounce 1

1. Modify the Flatland class so that the simulation runs for 1000 seconds.
2. Modify your Bot class so that the bots bounce when they hit the walls of Flatland.
    - Hint: your **tick** object will need to inspect the size argument of what's passed in.
    - If the object is going to exceed the bounds, you could just reverse the direction. (Can you do this in a line of code?)
    - Will this work for objects moving at angles?

RandomBot

Create a subclass of your Bot Class called RandomBot. This bot should randomly change direction every 10 seconds.

Now, how are you going to get a RandomBot onto the playing field? You'll do this by creating a

new simulation class called Simulation. Copy out the **main()** function in Flatland and create a Simulation.main(). This Simulation class should create a Flatland object, add a RandomBot to it, and then set the whole thing going.

Extra Credit

If you want, create a third class, add it to your simulation, and let it go. Here are some ideas:

- Have a bot that chases the RandomBot.
  - How will your Chaser instance find and pursue the RandomBot?
  - What will your chaser do when it finds the RandomBot?
- Right now the RandomBot will go through the other bots. How would you make it bounce off them?
- Give every object a mass. Compute the mutual attraction using Newton
  - How will you create the next step? (Hint: split the **tick()** method into two methods: **tick()** and **toc()**.

Deliverables

Your entire deliverable is a jar file containing Bot.java, Bot.class, RandomBot.java, RandomBot.class, Simulation.java, and Simulation.class. You can make this file with the *jar* command:

```
jar -cf Bot.jar Bot.java Bot.class RandomBot.java RandomBot.class Simulation.java
Simulation.class
```

You should upload it to our the class server. Hopefully there will be some kind of automated testing by Friday.

Notes

- All submissions must be in compliance with the CS3773_Java_Style_Guide (JavaDoc, Naming Conventions,

etc.)

- Your program should be robust: for example, inappropriate inputs should cause graceful failure.

See Also

Assignments

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

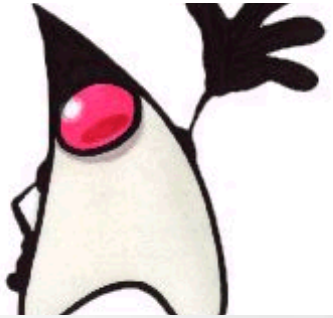
Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version

- Permanent link



Assignment 3

The goals of this week's assignment is to continue your exposure to object oriented design and programming.

We will be building upon the Flatland with two new simulations:

- Snake
- Electrostatics

The extra simulation will be:

- Shelling

Contents

Snake

In Week2 you created a Bot and RandomBot. In class we also created a ChaserBot.

For Week3 we will be creating yet another subclass of Bot called the AffinityBot. The AffinityBot will look in the world for bots that have a certain tag and will try to move closer to it.

We will start by adding **Tags** to the FlatlandObject. A tag is an integer attribute

that can be added to any object. In the original version of this problem set we suggested using a Tag interface, like this:

```
interface Tag {
   public int tag();
   public void setTag(int newTag);
}
```

And here is another interface, the Tag Search interface:

```
interface TagSearch {
   public ArrayList<TaggedFlatlandObject> search(int tag);
}
```

But this turned out to make the code really complicated, so we decided instead to just build the **Tag** functionality into the FlatlandObject abstract object, with these three methods:

```
    private int myTag=0;
    /** Tags are a way of keeping track of objects */
    public void setTag(int tag){
        this.myTag = tag;
    }
    public int tag(){
        return myTag;
    }
```

Either one works.

We have posted code with the new classes, which we will describe in class on Thursday. Here are the major changes:

1. Flatland now supports Tags.
2. We have modified the Flatland class to make the names more sensible.
3. We have modified the Flatland class to make it clear which methods and fields are private and which are public.
4. Flatland's **tick()** method no longer sends a **tick()** method to each of the objects; it now sends a **calculateNextPosition()** message then a **updatePosition()** message.
5. Flatland now includes a **run()** method which runs the simulation indefinitely.
6. The Bot() implementation has been cleaned up.
7. The ChaseBot() implementation is now given to you. It's pretty clean too.
8. The AffinityBot() implement is given to you, but you need to finish it off.

You are welcome to use your own classes, the classes from the original version of this assignment, or the final version of this assignment. You need to turn in a set of .class and .java files that makes the snake work, as demonstrated in class. You are welcome to add additional functionality as you wish. The only requirement for this part is that you have a snake, that it work, and that your code be beautiful.

Electrostatics

Rework your simulation so that there is just one kind of Bot, a ChargedBot bot.

- Every bot gets an electric charge, either +1 or -1.

- Give bots that have a +1 charge the color Blue, -1 charge the color Red.
- By the way --- if a +1 and a -1 charge get too close, they should stick but not fuse and become the same with exactly the same center. That is, if they both have a radius of 1, one should go to (1,0) and one go to (3,0), but both should not go to (2,0) and (2,0).
- Calculate each Bot's future position by calculating the force vector to every other bot and adding together all of the force vectors.
- Create an interesting initial configuration and watch what happens.

Schelling

Schelling is a simulation of self-segregation. It's played on an n x m grid. Only one object can be in each grid location at a time.

- There are Red bots and Blue Bots.
- Every clock tick, 10% of the Reds and 10% of the Blues decide to move to another grid location; of course, they can only move to an empty one.
- A Red bot is happy to live next to the same number of Reds and Blues, or is happy to live with more Reds. It would rather not live with more Blues.
- A Blue bot is happy to live next to the same number of Reds and Blues, but would rather not live with more Reds. (ie: same rules as the Blues, but reversed.)

Run the simulation. See what happens.

Hints on implementing Schelling:

- Have Flatland maintain a grid of [x][y] coordinates with a two-dimensional

array that is sized 50x50 or 100x100 (the same as your initial size).

- Go back to a single tick() method; it will be too hard to have every Bot figure out where they are moving and then all move at the same time. (Or have the bots calculate their move and move on the second update method.)
- Remember, this part is extra work, not necessary to turn in.

Check out Peter Wayner's schelling simulation: http://www.wayner.org/texts/seg/012298segregate-sim2.html

What to turn in

Turn in a jar file with your **username.jar**. The jar file should run each simulation when the simulation's name is provided as the first argument. eg:

```
java -classpath slgarfin.jar snake
java -classpath slgarfin.jar electrostatic
java -classpath slgarfin.jar schelling
```

See Also

- Assignments
- Week 3 Grading Rubrick

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

4

This week we are learning about input, output, and persistent data.

- We will dramatically simplify the simulation to avoid the threading problem that we've discovered.
- We will modify the so that the bot can take its commands from a file.
- You will run your simulation.
- You will then modify your simulation so that the commands can be taken from an SQL database over the network.
- We will develop this simulation over the week in class, learning about exceptions and remote database access.

And one last thing:

- We will start using packages. In this case, package hw4.

Contents

Problem 1

Download these two Java files to your computer:

- https://domex.nps.edu/cs3773/svn/week4/B.java
- https://domex.nps.edu/cs3773/svn/week4/BTest.java

Here is BTest.java:

```
package hw4;
public class BTest {
    public static void main(String[] args){
        B b = new B(1);
        System.out.printf("The value of the B function at 1 is: %d %n",b.calc());
    }
}
```

When BTest.main is run, the program prints out the value of the B function when calculated at 1.

Add a new static method to BTest called number_of_exceptions(int n) which returns the number of exceptions that the B function generates when evaluated between 1 and n. The pseudocode for this function looks something like this:

```
    public int number_of_exceptions(int n){
       for(int i=1;i<=n;i++){
           // some sort of try here
           B b = new B(i);
           b.calc();
           // some sort of catch here
       }
       return the number of exceptions that happened;
     }
```

So the following values should hold:

```
  BTest.number_of_exceptions(1) should return 0
  BTest.number_of_exceptions(2) should return 0
```

```
  BTest.number_of_exceptions(3) should return 1
```

Modify BTest.main so that it prints the number of exceptions generated between 1 and 100 using

your static method number_of_exceptions()

Problem 2 - Throw an exception

Download this Java file to your computer:

```
   https://domex.nps.edu/cs3773/svn/week4/CTest.java
   https://domex.nps.edu/cs3773/svn/week4/CInterface.java
   https://domex.nps.edu/cs3773/svn/week4/CException.java
```

CTest.java wants a class called C which implements the CInterface.

Create C.java on your computer. It's going to look a bit like this:

```
public class C implements CInterface {
    // todo: add your code here
}
```

There is also

Your C class needs to implement a single method called calc(int). Here are the rules for the C.calc function:

1. C.calc(n) for any even number should throw an `ArithmeticException`.
2. C.calc(99) should throw a CException with the exception's value set to be 33.
3. Otherwise, return n.

Both of these are tested by the CTest.java

Test this with:

```
java CTest
```

Problem 3 - Create a File

Here is a program that puts "Hello World!" into a file called output.txt:

```
package hw4;
import java.io.*;
public class HelloToFile {
    public static void main(String[] args){
        try{
            PrintWriter out = new PrintWriter(new FileWriter("output.txt"));
            out.println("Hello World!");
            out.close();
        } catch (IOException e){
            System.out.println("Could not open output.txt");
            e.printStackTrace();
        }
    }
}
```

Because this class is called HelloToFile it should be stored in a file called HelloToFile.java.

1. Add a public static method to this file with the following signature:

```
    public static void makeHelloFile(String filename) throws IOException;
```

When `makeHelloFile` is called, it should put 'Hi Mom!' (followed by a newline) into the file named by **filename**.

2. Modify the **main** static method so that, instead of putting **Hello World!** into the file called **output.txt**, the program calls the `makeHelloFile` method 10 times with the filenames *file*1.txt through *file*10.txt where *file* is specified on the command line. That is, this invocation should create the

files foobar1.txt, foobar2.txt, etc.:

```
java HelloToFile foobar
```

And this should create nosmis1.txt, nosmis2.txt, etc:

```
java HelloToFile nosmis
```

note -- if your main method encounters an exception on file N, it should continue with file N+1.

You may find the following web pages helpful in performing this question:

- Java PrintWriter documentation
- Java FileWriter documentation
- Java Console and File Input/Output Cheat Sheet
- PowerPoint deck on file I/o

If you want, you may also read:

- Day 15, Working with Input and Output, in Teach Yourself Java 6 in 21 days.

Problem 4 - Download a web page

Here is some code that will not compile. The code should download a web page called http://domex.nps.edu/cs3773/hello.txt and print it line-by-line. This code introduces three classes you have not seen before: **InputStream**, **InputStreamReader**, and **BufferedReader**.

```
package hw4;

import java.io.*;
import java.net.*;

```

```
public class WebCalc {
    static public void main(String[] args){
        URL u = new URL("http://domex.nps.edu/cs3773/hello.txt");
        URLConnection con = u.openConnection();
        if(con instanceof HttpURLConnection){
            int code = ((HttpURLConnection) con).getResponseCode();
            if(code!=HttpURLConnection.HTTP_OK){
                System.out.println("Error code: "+code);
                return;
            }
        }
        InputStream is = con.getInputStream();
        InputStreamReader isr = new InputStreamReader(is);
        BufferedReader br = new BufferedReader(isr);
        while(true){
            String s = br.readLine();
            if(s==null) break;
            System.out.println(s);
        }
        br.close();
        is.close();
    }
}
```

1. Fix this code so that it compiles. Your output should look like this:

```
$ java hw4.WebCalc
Hello World!
```

2. Modify this program so that, if an argument is provided, the provided argument is treated as a URL

and that page is downloaded and printed. Your output should look like this:

```
$ java hw4.WebCalc http://domex.nps.edu/cs3773/mom.txt
Hi Mom!
```

3. Modify the program so that a "-sum" option before the URL causes the program to add up all of the numbers on the web page (assuming each number is on its own line). Here are two sample outputs you should get:

```
$ java hw4.WebCalc -sum http://domex.nps.edu/cs3773/numbers.txt
6


$ java hw4.WebCalc -sum http://domex.nps.edu/cs3773/numbers2.txt
line 4: "foobar" is not a number.
```

Problem 5 - The Web Crawler

This problem is extra credit...

Create a new class called WebCrawler.

Give it a public static method called crawl(String url) which does the following:

- Opens up the url **url**
- Reads each line
- Prints the line
- If the line contains a URL, calls **crawl(url)** with the URL on the line.
- Returns the contents of the web page
- Increments a static instance variable keeping track of the total number of lines printed and URLs visited.

Create a public static method called main(String args[]) which:

- Calls crawl("http://domex.nps.edu/cs3773/hw4/one.txt")
- Prints the total number of pages printed and exits.

```
Total lines: 29 total URLs: 6
```

This is really fun. If you can get it to work, try to add the following:

- A global variable that keeps track of each page visited and doesn't visit the same page twice.

  (Very Important!!! You can test this with the url http://domex.nps.edu/cs3773/hw4/here.txt which points to http://domex.nps.edu/cs3773/hw4/there.txt and vice-versa).

  - Hint: You want to use a Hashtable which maps the string URLs to a boolean flag.
  - Better Hint: Use a HashSet instead of a Hastable. Declare it as HashSet<String>; put strings into it and then see if you were at the page by seeing if the HashSet contains the specified key.

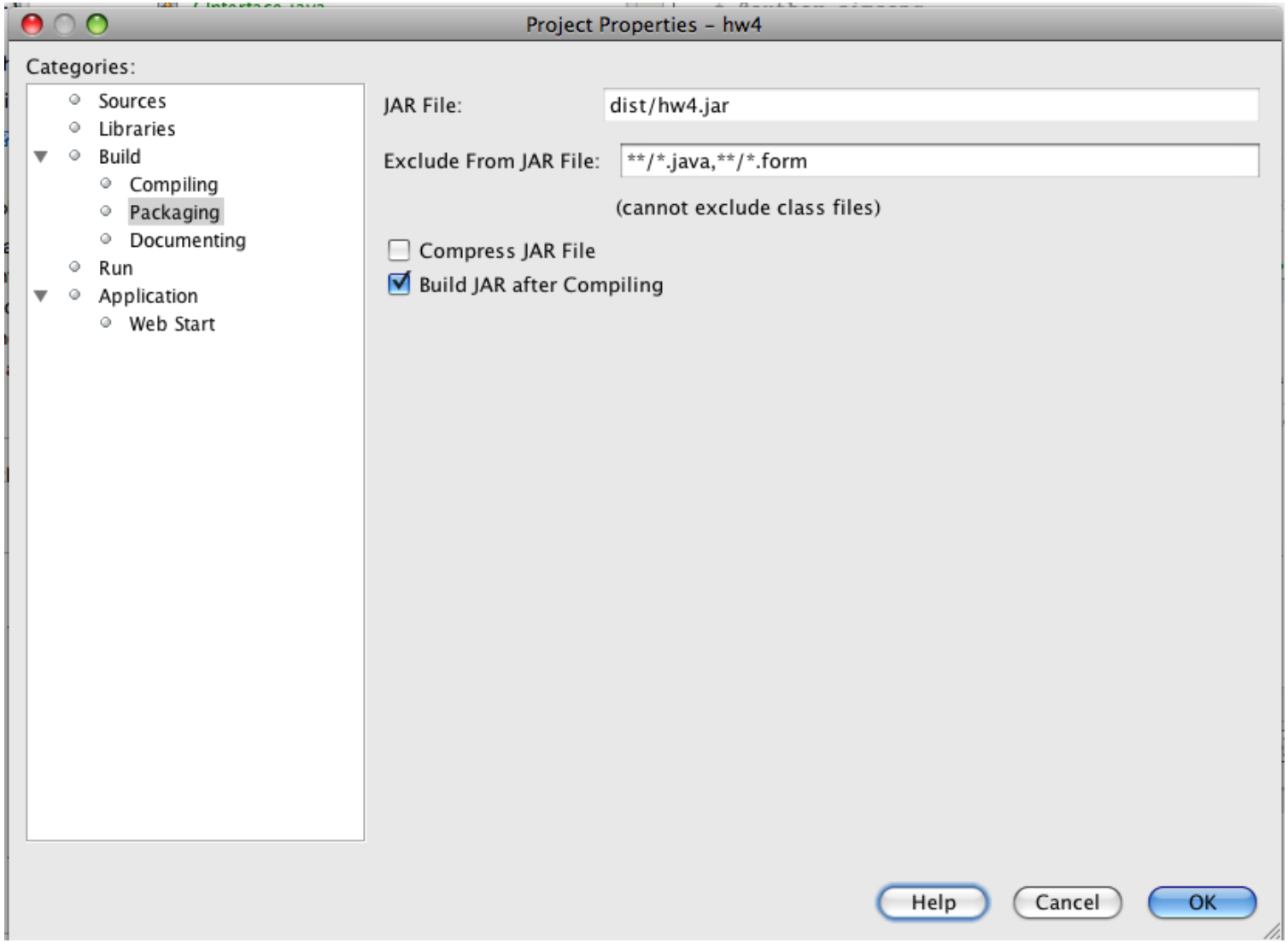
- A global counter which keeps track of the total number of pages visted; print this when you are done.
- Try printing the pages completely, rather than line by line. Hint: do this with a StringBuilder.
- Question: how could you make your program work with actual HTML rather than with these text files?

HINT

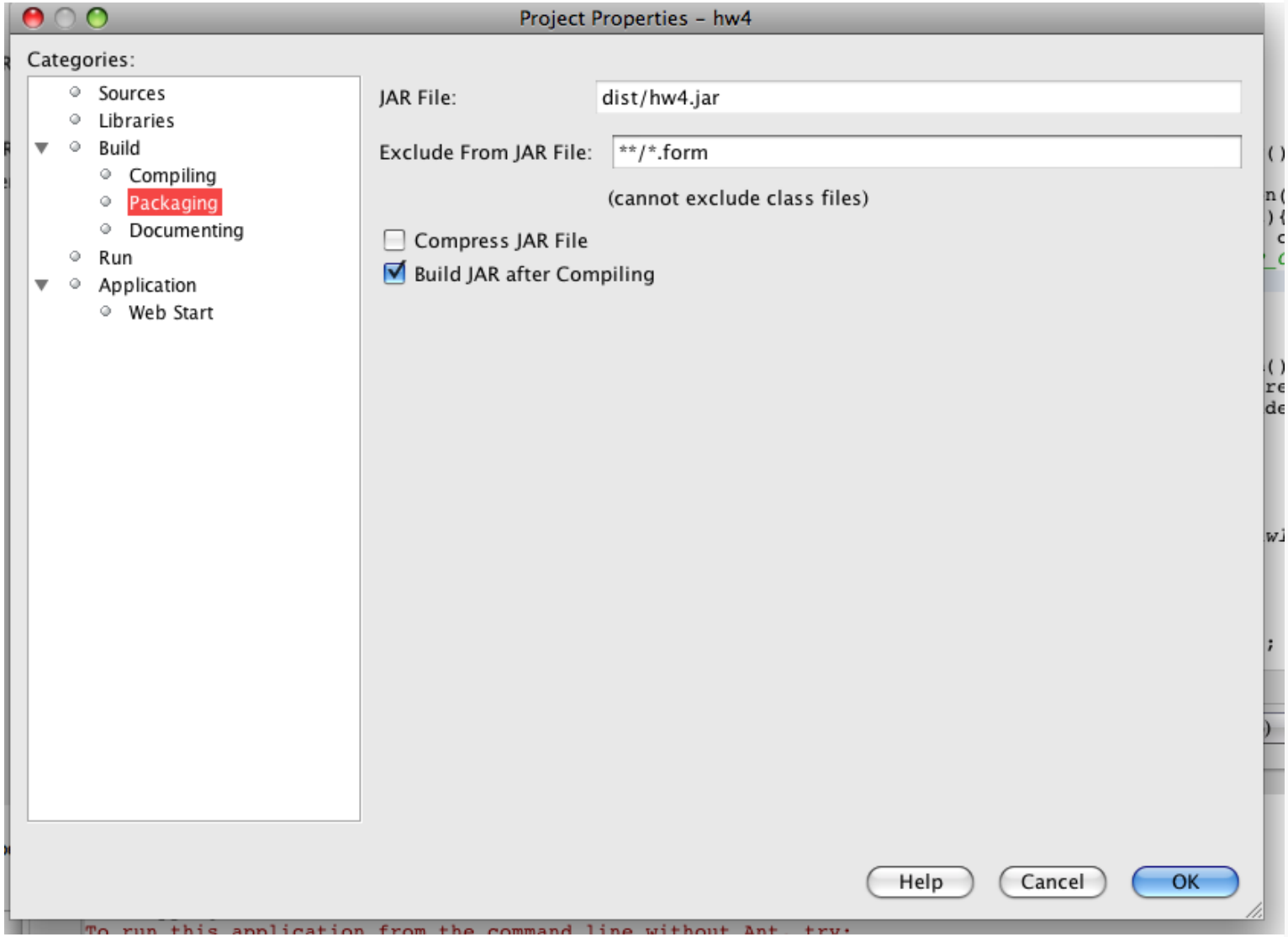
If you are using NetBeans, you can keep track of each of these different commands with the run profiles.

If you tell NetBeans not to exclude the *.java files, it will automatically include them.

Change this:


Project Properties - hw4
Categories:
Sources
Libraries
Build
Compiling
Packaging
Documenting
Run
Application
Web Start
JAR File:
dist/hw4.jar
Exclude From JAR File:
/*.java,/*.form
(cannot exclude class files)
Compress JAR File
Build JAR after Compiling
Help
Cancel
OK

to this:



What to turn in

Turn in a jar file containing at least the following:

```
hw4/BTest.java
hw4/BTest.class
hw4/C.java
hw4/C.class
hw4/HelloToFile.java
```

```
hw4/HelloToFile.class
hw4/WebCalc.java
hw4/WebCalc.class
hw4/WebCrawler.java
hw4/WebCrawler.class
```

- Homework Submission Link.

Grading=

Problems 1-4 are worth 25 points each; Problem 5 is worth 20.

See Also

- The exceptions debate

Assignments

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

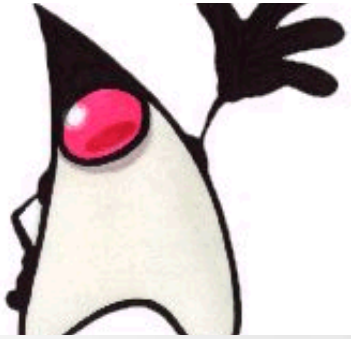
Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file

- Special pages
- Printable version
- Permanent link



Assignment 5

This week's assignment involves a code-cleaning exercise and extensions to the Flatland GUI. You are to submit a jar file, but the assignment will not be graded automatically.

Part 1: Newton

In the week5 subversion repository you will find two fractal programs: Mandelbrot and Newton.

- Download a copy of the subversion repository from http://domex.nps.edu/cs3773/svn/
- Fix Newton.java. Turn it into beautiful code. Be especially aware of:
  - Dupliate code that can be cleaned up.
  - Variable names that can be clarified.
  - Dead code that can be removed (is there any?)
  - Hard-coded constants that should be clarified.
  - Comments that should be improved.
- At the beginning of Newton.java, put a summary of the code cleaning you did.

Plan on spending 2-3 hours cleaning this up.

Part 2: Flatland GUI

In the week5 subversion repository you will find a reworked version of Flatland. This version uses just a single thread and, as a result, it does not throw the exception that troubled is in Week 3. This version also implements the clean architecture that we discussed in the homework.

Study the code that you have been given. Add at least four of the following features:

- Buttons to start and stop the simulation
- Rework the simulation layout and make it more attractive.
- Make the simulation run in a JApplet instead of a JFrame
- Add walls to the inside simulation which the random objects bounce off
- Let the user draw new walls with the mouse.
- Let the user select a bot with the mouse and:
  - Control it with the keyboard
  - Watch where it moves (show its trail)
  - Change its properties by right-clicking on it.
- Make the bots bigger and draw the bot's tag in the middle of the bot.
- Draw a line between the ChaseBot and its target.
- Annotate the target line with the length.

Category: Assignments

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

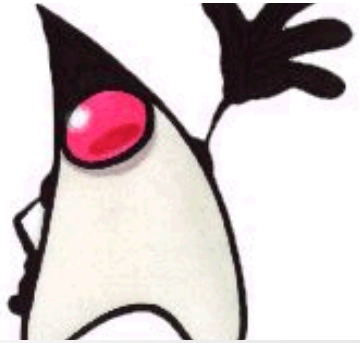
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Assignment 6

Assignment #6 will be due on **Monday, February 18th.**

Re-implement the Flatland from HW5. This time create the FlatlandControlPanel using the NetBeans GUI builder.

Extra credit: Add and document some new functionality. Here are some ideas:

- A bot you can steer
- A bot that creates new bots.
- A bot that destroys bots.
- A bot that changes colors
- A bot that displays an image

CS 3773

- Main Page
- Week-by-week Outline
- Assignments

- Readings

Wiki

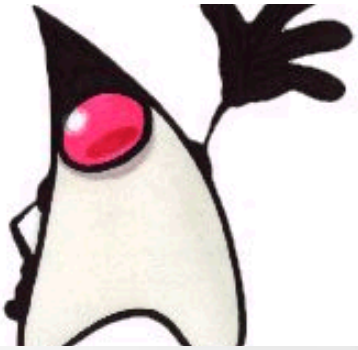
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Assignment 7

Assignment 7 is optional. Please make your Flatland run as an applet.

When you upload Assignment 7, you will be given a URL for your applet.

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

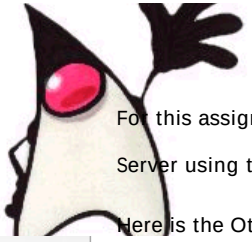
Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version

- Permanent link



nent 8

For this assignment, you are to create a simple application that communicates with the Othello Server using the XML RPC API.

Here is the Othello Server:

- http://othello.nitroba.org/game.cgi

Here are some code snippets:

- joinGameButtonActionPerformed

We will discuss this API in greater depth in class and sample code will be provided by Wednesday.

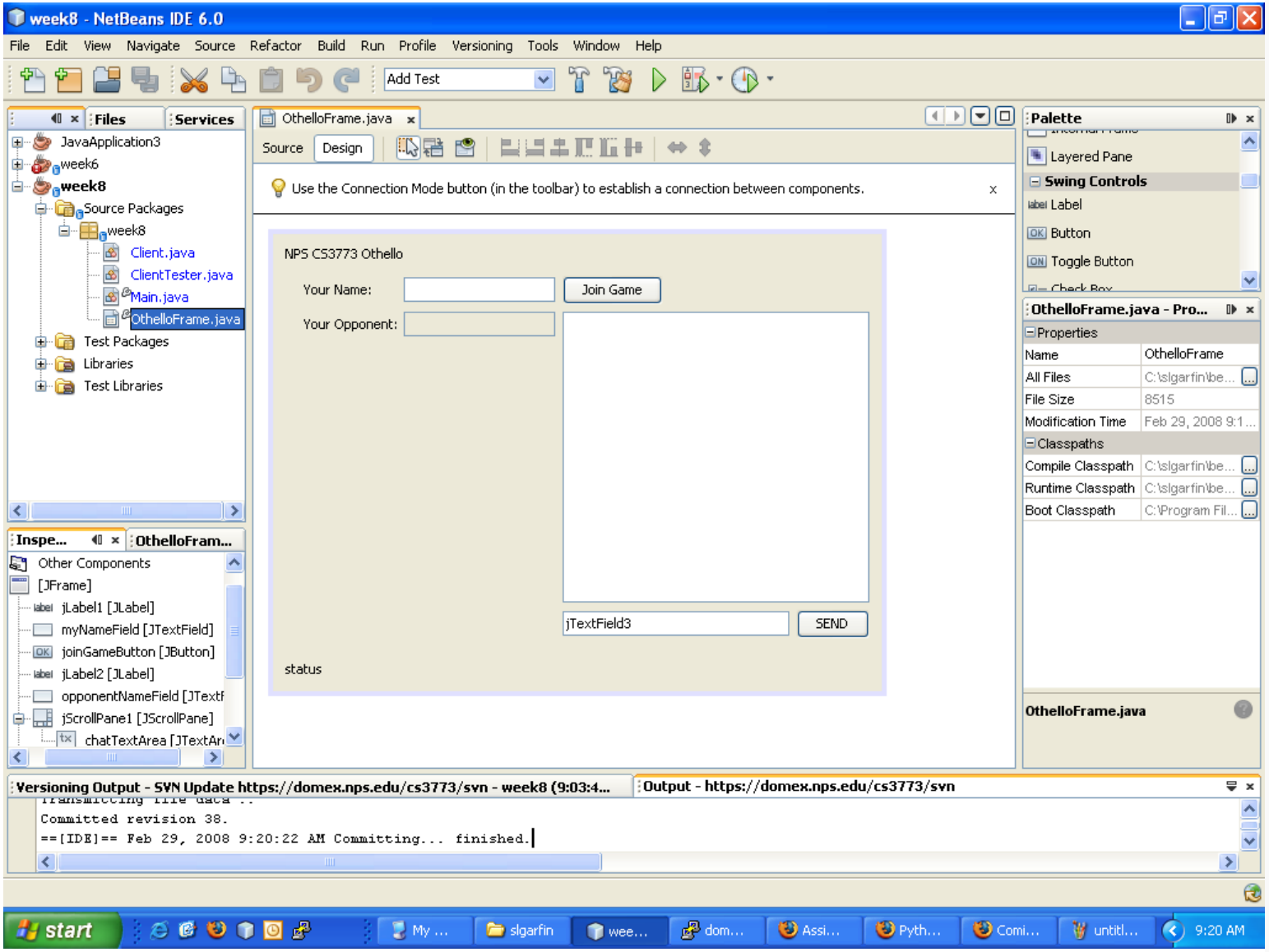
Here is how you implement the chat program:

- Join a game by calling joinGame() with your name as the playerID
- This will give you a gameID, which is an integer
- Every second, you must call stillAlive(playerID,gameID) to tell the server that you are still alive.
- Every second, you should call getMessages(gameID,playerID) to see if there are any messages for you. If there are, display them.
- Every second, you should call getGameState(gameID) to find out if someone else has joined the game, and to find out if you are white or black, and to find out if it is your turn to move.
- If you want to send a message to someone, call sendMessage(gameID, senderID, recipientID, message)

To implement a full-blown game, you would need to do this:

- Display the game board after getGameState() is called
- If it is your turn, pick up a mouse click on the game board and send it to the server as a submitMove() function call.

Here is a sample screenshot:



Assignment #8 is due on Friday, March 7th.

Grading will be as follows:

- 50% credit - turning in a jar file that connects to the othello server and displays the current status (updated every second) in a text window.
- 25% credit - connects to the server, joins a game, and displays chat messages as they are received.
- 25% credit - allows the user to send chat messages to the other user.
- 25% extra credit - displays the blank othello board.

- 25% extra credit - lets you click on board positions and send the moves
- 25% extra credit - gets boards from the othello server and displays them
- 25% extra credit - properly handles game over.

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Final Project

Final Projects are due on March 27th

Your final project must include:

- Final Project Proposal
- Java source code that is **well-written** and **well-documented.**
- A written specification of at least one page that describes:
  - The program's purpose.
  - The program's interface with the outside world.
  - The program's theory-of-operation.
  - What is original, and what (if anything) was developed by others.
  - Future work that can be done.
- Screen shots.

To Create Your Final Project Proposal

If you wish, you may put your final project proposals on this wiki. If you would prefer not to, you may email the final project proposal to the course staff.

To enter the proposal on the wiki:

1. Type your name and the words "final project proposal" into the Search box at the left of this window.
2. MediaWiki will ask if you want to create a new page; click "yes"
3. Enter your proposal.

4. When you are done, please click "save page"

Note: You will be responsible for making sure that your final project matches what was promised in the proposal. If you put your proposal on the wiki it will be easier for you to keep the proposal up to date as the project changes.

What to turn in

- A jar file (please submit using the online system and e-mail a copy to the professor)
- A PDF file with your specification and screen shots (please upload separately and email to me if you can't get that to work.)

See Also

Final Project Ideas

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

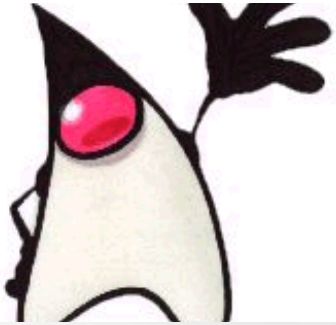
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Week 01: Java Without Classes

- **Next:** Week 02: Classes

	Introduction
Week	1
shortname	Intro

Contents

Goals

- Course Goals
- Understand Java and how it's different from C++
- Compile Hello World
- Turn in your first assignment.

Outline

Administrivia

- Course Schedule
- Weekly Schedule:
  - Mondays - New Topic is introduced; slides
  - Tuesdays - Delve deeper into new topic; code
  - Wednesdays - Java Article to discuss; Q&A
  - Thursday - Corner cases; more about the topics.
  - Friday - Lab. Mini quiz; Q&A; homework must be submitted by 1800 Sunday (6pm) Pacific Time.
- Survey: What do you know? What do you want to get out of the course?
- How to use the wiki
- How to use Subversion to get the course material.

Welcome to Java

- Busy Box - be amazed

- Java pros/cons
- History of Java

Hello World!

Here is a simple Java program:

```
public class hello {
    public static void main(String args[]){
        System.out.println("Hello World!");
    }
}
```

This program *must* be put in a file called **hello.java**.

You compile this program with the **javac** compiler:

```
% javac hello.java
```

You run this program with the **java** command:

```
% java hello
Hello World!
```

Other things that we'll teach today:

- Using the Java online documentation.

- Installing the Java system on your computer.
- What CLASSPATH does
- applets vs. stand-alone programs.
- GUI vs. command-line programs.

Writing Java

- Compiling and running from the command line
- IDEs - emacs, BlueJ, NetBeans, Eclipse,
- We will be using Eclipse

Running Java

- Windows & Unix
- CLASSPATH
- PATH, JAVA_HOME, and Classpath
- C:\Java\j2sdk1.4.2\jre\lib\ext

Java Documentation

- How to find the documentation.
- Javadoc

Introduction to Java Sytax

- math (integer; float, etc)
- Data types & operators
- Using strings, ints, arrays
- Control flow - for (c-kind), if/then/else, switch, while, for (python kind)

- import and packages
- operator precedence

Java vs. C++

What's mostly the same:

- basic types (except for string)
- syntax
- classes
- System.output.println vs. cout <<

What's different:

- Single inheritance
- string type
- no objects on the stack (no automatic object creation)
- no operator overloading
- JavaDoc
- bytecode vs. object code
- anonymous classes

Java vs Python

What's mostly the same:

- Single inheritance
- bytecode, not object code

- basic types (including string)

What's different:

- syntax
- classes
- no objects on the stack
- no operator overloading
- JavaDoc

Slides

- Slides for Week 1 Monday, Tuesday
- Wednesday
- Thursday

Readings

- Read the CS3773 Java Style Guide.
- Teach Yourself Java 6 in 21 Days. Please review chapters 1 through 5 by the end of the week.
- Wednesday Paper: Enhancing the introductory computer science curriculum: C++ or Java? Andrei Irimia, CCSC 2001.

Assignments

Assignment 1 is due on Sunday, January 13, at 6pm pacific time

---

- **Next:** Week 02: Classes

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

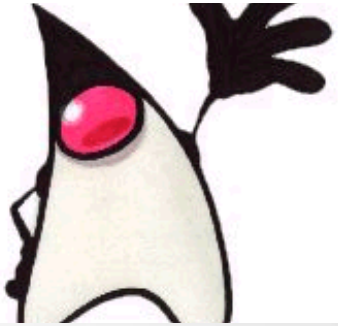
Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version

- Permanent link



Week 02: Classes

- **Previous:** Week 01: Java Without Classes
- **Next:** Week 03: Object-Oriented Design

	Classes
Week	2
shortname	Classes

Contents

Goals

Language Goals

- What is a class? What is an instance? What is an object?
- Understanding inheritance: methods & variables
- Encapsulation: public variables vs. accessor methods
- Final: final methods & final variables
- mutable vs. immutable objects
- Enums

Environment Goals

- applets vs. stand-alone programs.
- GUI vs. command-line programs.
- What CLASSPATH does
- Using the debugger

Outline

Monday: Basics of Classes and Object-Oriented Design

- Monday Slides

Tuesday: Classes With Code

- Show the Demo
  - Download week2.jar

    - Create a Bot.java
    - `java -classpath week2.jar;. Flandland` (on Windows)
- Show JavaDoc for week2

New Java reserved keywords for working with a single class:

public

New Java reserved keywords for working with class hierarchies:

abstract

extends

implements

instanceof

interface

protected

Some code to discuss

Readings

Last week we asked you to review Chapters 1-5 in *Teach Yourself Java 6...*

This week we be working through the book.

For Tuesday

- **Day 3,** *Teach Yourself Java 6 in 21 Days*. This chapter discusses **new**, methods, method calls, nested method calls, casting, and the **instance of**

operator.

- Review the Q&A on p. 112. You should know the answer to all of these questions.
- Take the quiz on p. 112--113.
- Review the Q&A on p. 85. Be sure you can answer all of the questions.
- Take the Quiz on p. 86.
- If they look hard, do the exercises on p. 87 (for yourself, not to hand in).

For Wednesday

(But you may wish to do it in advance of Wednesday):

- Java Pitfalls for Beginners , Robert Biddle and Ewan Tempero, SIGCSE, June 1998
- **Day 5,** *Teach Yourself Java 6 in 21 Days.* This chapter discusses **new**, methods, method calls, nested method calls, casting, and the **instance of** operator.

For Thursday

- **Appendix A: Using the Java Development Kit,** *Teach Yourself Java 6 in 21 Days.* This chapter discusses **new**, methods, method calls, nested method calls, casting, and the **instance of** operator.

Class Notes

Week 02 Thursday Notes

Navigation

- **Previous:** Week 01: Java Without Classes
- **Next:** Week 03: Object-Oriented Design

Category: JavaWeek

- Upload file
- Special pages
- Printable version
- Permanent link

ject-Oriented Design

	Object-Oriented Design
Week	3
shortname	OOP

- **Previous:** Week 02: Classes
- **Next:** Week 04: I/O

Contents

Goals

- Inheritance
- Interfaces vs. Abstract classes
- Nested Classes
- Enumerated Types
- overriden methods

- overloaded methods
- overriding vs. overloading
- packages
- Annotations

Class Notes

Finishing up from last week

super constructors

To call the the constructor in the super-class, use this syntax:

```
  super();
```

So the ChaseBot implementation that we presented in class on Friday is more properly implemented as this:

```
import java.lang.Math.*;

public class ChaseBot extends Bot {
    public ChaseBot(Location center,Size size) {
        super(center,size);
    }

    public void tick(Flatland fl){

        /* Find the random bot */
        RandomBot target = null;        // the random bot that we are looking
```

```
for
        for (FlatlandObject o : fl.things ){
            if(o instanceof RandomBot) target=(RandomBot)o;
        }
        if(target==null) return;               // don't move if there is no
target

        /* First calculate the values for what this Bot will look like in 1
second... */
        double dx = target.center.x-this.center.x;
        double dy = -(target.center.y-this.center.y);

        /* Set the new Heading with the atan2 function */
        heading = Math.toDegrees(Math.atan2(dx,dy));

        super.tick(fl);                // and
move
    }
}
```

And the RandomBot looks like this:

```
import java.lang.Math.*;

public class RandomBot extends Bot {
    int duration = 0;

    public RandomBot(Location center,Size size) {
        super(center,size);
    }
    public void tick(Flatland fl){

        if(duration<=0){
```

```
            duration = (int)(Math.random()*100.0+30.0);
            heading = Math.random()*360;
        }
        duration = duration - 1;
        super.tick(fl);
    }
}
```

Get all my code: http://domex.nps.edu/cs3773/svn/week2/

default constructors

If you don't create one, Java creates one for you:

```
public class Default {
    public print() {
        System.out.println("Hello!");
    }
}

...

Default d = new Default();
d.print();
```

BotTester vs. BotTester2

BotTester implemented **Regression Testing**

BotTester2 implemented **Conformance Testing** or **Unit Testing**

Regression Testing:

- Run the same code with the same inputs and see if the output changes.
- Code 1 was slgarfin.jar
- Code 2 was **yourname**.jar

Conformance Testing:

- Run the code with a '*test harness* and check the outputs.

Regression Testing Advantages:

- Easy to implement.
- Great for test new versions of code for release.

Regression Testing Disadvantages:

- May not find bugs.

Conformance Testing Advantages:

Conformance Testing Disadvantages:

Common Mistakes on the Homework

- Duplicated code between Bot and RandomBot
- Hard-coded constants.

- Comments that weren't relevant.

This is useless:

```
// publicRandomBot() creates a RandomBot():
public RandomBot(Location center, Size s){
  ...
```

This is almost useless:

```
// Decrement counter
counter -= 1;
```

Try this:

```
// When counter reaches 0, it's time to change direction:
counter -= 1;
if(counter<=0){
   ...
```

What is "Business Logic?"

- Model/View/Controller [See http://en.wikipedia.org/wiki/Model-view-controller]
- Three-Tier Architecture

Abstract Classes vs. Interfaces

Flatland uses both an Abstract class and an Interface.

- Abstract Class: FlatlandObject
- Interface: FlatlandDelegate

Advantages of Abstract Classes:

Advantages of Interfaces:

Java Packages

- Syntax
- What's in the package
- import

Java Scoping Rules

- local scope
- package scope
- protected scape
- public scope

See http://mindprod.com/jgloss/scope.html

public

any class can reference it

protected

only classes that extend this class can reference it.

default (package; friendly)

any method in this package can access it.

private

no class can reference it

local

only visible in the block.

```
package one;
public class A {
    protected int b;
    void public scope_demo() {
        int local_scope = 3;
    }
}

package two;
import one.A;
class B extends A {
    void myMethod() {
        p = 1;  // ok
        A a = new A();
        a.p = 1; // not okay; p would have to be public
    }
}
```

Example from http://mindprod.com/jgloss/protectedscope.html

-cp vs -classpath

A question was asked in class about -cp vs. -classpath. Apparently they both work. See http://javahowto.blogspot.com/2006/06/new-options-in-javac-155-classpath-cp.html .

Nested Classes

Java 1.5 and above allows you to create classes inside a class. This is useful if you wish to create an internal class which no other class should be able to use.

Here is a somewhat trivial example:

```
public class Test {
    private class InnerClass {
        StringBuilder res = new StringBuilder();
        String appendFoo(int i){
            while(i-- > 0){
                res.append("foo ");
            }
            return res.toString();
        }
    }

    public void run(){
        System.out.println("let's run the foo function...");

        InnerClass ic = new InnerClass();
        System.out.println("foo(3)="+ic.appendFoo(3));
        System.out.println("foo(2)="+ic.appendFoo(2));
        System.out.println("foo(1)="+ic.appendFoo(1));
    }


    public static void main(String[] args){
        Test t = new Test();
        t.run();
    }
}
```

```
```

The InnerClass is **private,** so it can only be accessed from within the **Test** class.

Here is what the output looks like:

```
10:27 PM imac2:~$ javac Test.java
10:27 PM imac2:~$ java Test
let's run the foo function...
foo(3)=foo foo foo
foo(2)=foo foo foo foo foo
foo(1)=foo foo foo foo foo foo
10:27 PM imac2:~$
```

Don't worry if this seems complicated right now.

Enumerated Types

Added in JDK 1.5.

See http://java.sun.com/docs/books/tutorial/java/javaOO/enum.html

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Points of note:

- Enumerated types are types, like classes. You could put the **enum Day** into a file called **Day.java**

and compile it with javac.

- Enumerated types are really syntactic sugar. They don't exist at the bytecode level.

Here is how the Day.class file decompiles:

```
10:53 PM imac2:~$ jad Day.class
Parsing Day.class... Generating Day.jad
10:53 PM imac2:~$ more Day.jad
// Decompiled by Jad v1.5.8g. Copyright 2001 Pavel Kouznetsov.
// Jad home page: http://www.kpdus.com/jad.html
// Decompiler options: packimports(3)
// Source File Name:   Day.java


public final class Day extends Enum
{

    public static final Day[] values()
    {
        return (Day[])$VALUES.clone();
    }

    public static Day valueOf(String s)
    {
        return (Day)Enum.valueOf(Day, s);
    }

    private Day(String s, int i)
    {
        super(s, i);
    }

    public static final Day SUNDAY;
```

```
    public static final Day MONDAY;
    public static final Day TUESDAY;
    public static final Day WEDNESDAY;
    public static final Day THURSDAY;
    public static final Day FRIDAY;
    public static final Day SATURDAY;
    private static final Day $VALUES[];

    static
    {
        SUNDAY = new Day("SUNDAY", 0);
        MONDAY = new Day("MONDAY", 1);
        TUESDAY = new Day("TUESDAY", 2);
        WEDNESDAY = new Day("WEDNESDAY", 3);
        THURSDAY = new Day("THURSDAY", 4);
        FRIDAY = new Day("FRIDAY", 5);
        SATURDAY = new Day("SATURDAY", 6);
        $VALUES = (new Day[] {
            SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
        });
    }
}
```

Several students asked *what is a enumerated type good for?*

Enumerated types are useful when you have a variable that you want to restrict to particular values, but it doesn't make sense to have a different class defined for each variable type.

Here are examples:

- Days of the week.
- Months

- Planets
- Card suits (clubs, diamonds, hearts, spades)

References

- HSQLDB, a relational database written entirely in Java.
- Java Tutorial on packages
- Java Tutorial on Annotations
- equals

Wisdom from Effective Java

The book *Effective Java* (Bloch, 2001) is the very best book on Java that I've found. Everything I tell you that is at odds with the book is wrong. Really, this book is amazing.

The books substance is 57 **items** which describe good, effective, clear and efficient programming style in Java. I'll list them here as we've covered enough about Java for them to make sense. Where the items are in odds with something I've done or said, I'll let you know.

Item 1: Consider providing static factory methods instead of constructors

The reason for this is that a class can have multiple static factory methods, but only one constructor with a signature. This has already bitten us in the Flatland example; I should have done things differently, it seems. Another advantage of this approach is that the static constructors can cache objects that are created and return multiple references to the same object, especially if the objects are immutable. So I should have followed this advance with the Size() and Location classes. If we have time, I'll show you how.

Item 4: Avoid creating duplicate objects

Don't do this:

```
String s = new String("silly");
```

do this:

```
String s = "No longer silly";
```

Item 5: Eliminate obsolete objects references

We haven't come up against this one yet. But when you don't need an object anymore, overwrite its reference with **null** to force garbage collection.

Item 13: Favor immutability

We do this with our Location and Size classes.

Item 14: Favor composition over inheritance

We are doing this too. It's usually better to have an object that contains other *helper* objects, rather than to try to bundle everything into parent classes.

Item 15: Design and document for inheritance or else prohibit it

Item 16: Prefer interfaces to abstract classes

Item 17: Use interfaces only to define types

Item 19: Replaces structures with classes

Bloch recommends avoiding this:

```
class Point {
  public float x;
  public float y;
}
```

and instead doing this:

```
class Point {
  private float x;
  private float y;

  public Point(float x, float y) {
    this.x = x;
    this.y = y;
  }

  public float getX() { return x; }
  public float getY() { return y; }

  public void setX(float x) { this.x = x; }
  public void setY(float y) { this.y = y; }
}
```

Did you notice that Bloch violates his own rules about immutability and static factory members?

Item 20: Replace unions with class hierarchies

Readings

- Day 6, Packages, Interfaces, and Other Class Features, *Teach Yourself Java in 21 Days,*
  - Be sure you understand the Q&A on p. 178.
  - Take the Quiz on p. 179
  - Answer the Certification Practice on pp. 179-180
  - Do the exercises on p. 181 (not graded)
- Read the Java Tutorials Lesson on Packages
  - Do the Java Tutorial Questions and Exercises on Packages]
- Taming the Tiger: Teaching the Next Version of Java™ , Jeremy D. Frens, SIGCSE'04

- **Previous:** Week 02: Classes
- **Next:** Week 04: I/O

Category: JavaWeek

- Week-by-week Outline
- Assignments
- Readings

Wiki

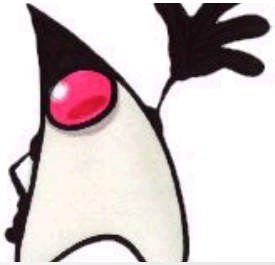
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Week 04: I/O

- **Previous:** Week 03: Object-Oriented Design
- **Next:** Week 05: Basic Graphics and GUI with AWT and Swing

- Week 3 Redux

	I/O
Week	4
shortname	I/O

Contents

Goals

- Exceptions - try/catch/finally/throws
- I/O Exceptions
- Keyboard input
- Streams, Buffered Streams
- Files
- Persistance
- SQL/MySQL/JDBC

Notes

This is a draft right now; it should be ready by Monday morning.

Null reference exceptions

Many students have been encountering errors that look like this:

```
Exception in thread "main" java.lang.NullPointerException
        at ChaseBot.rangeToTarget(ChaseBot.java:21)
        at A.main(A.java:5)
```

This is a stack trace. You read it from the top down:

Exception in thread "main" java.lang.NullPointerException

> This means that there was a Null Pointer Exception, which means that there was an attempt to reference a method or field of a pointer, but the pointer pointed to *null* .

at ChaseBot.rangeToTarget(ChaseBot.java 21)

This tells you **where** the exception took place: the *rangeToTarget* method in the *ChaseBot* class, and specifically at line *21* inside the file (which is going to be the ChaseBot.java file)

at A.main(A.java 5)

This tells you where the call was made to the *ChaseBot.rangetoTarget()* method. The call was made in the method *A.main*, specifically line 7 of the file *A.java.*

Let's look at the ChaseBot.rangeToTarget method. The numbers on the left are line numbers:

```
20    public double rangeToTarget(){
21      double dx = target.getCenter().x-this.getCenter().x;
22      double dy = -(target.getCenter().y-this.getCenter().y);
23      return Math.sqrt(dx*dx + dy*dy);
24    }
```

In line 21, you'll see that there is an attempt to run the *getCenter()* method of *target* and of *this* . The variable *this* is always set (it's always the object which has received the method), but the instance variable *target* is only set if somebody has called the object's *setTarget()* method.

Now it's time to look at the file A.java. This is test program that was constructed to demonstrate the exception:

```
public class A {
    public static void main(String[] args){
        ChaseBot b1 = new ChaseBot(new Location(10,10),1.0);
        ChaseBot b2 = new ChaseBot(new Location(20,20),1.0);
        System.out.printf("b1 range to b2: %g %n",b1.rangeToTarget());
    }
}
```

So what's clearly happened is that we've asked b1 to calculate its range to the target, but we never gave it a target. So it's thrown a null reference exception. The fix is easy: just set the target!

```
    public static void main(String[] args){
```

```
        ChaseBot b1 = new ChaseBot(new Location(10,10),1.0);
        ChaseBot b2 = new ChaseBot(new Location(20,20),1.0);
        b1.setTarget(b2);
        System.out.printf("b1 range to b2: %g %n",b1.rangeToTarget());
    }
```

And now we'll get the answer:

```
$ java A
b1 range to b2: 14.1421
$
```

Uses of Java Exceptions

Exceptions don't have to make your program crash: Java gives us tools to allow programs to catch exceptions and do intelligent error recovery. This allows your program to deal with *exceptional* cases, rather than simply crashing.

All programming languages have a strategy for dealing with runtime errors. In C the traditional approach is to look at the *return codes* of system calls. For example, if you want to open a file in C, you use code like this:

```
  File *f = fopen("filename.txt","r");
```

The C documentation says that you need to examine the return code of "fopen" to see if the file is properly opened or not:

```
  File *f = fopen("filename.txt","r");
  if(f==0) {
    printf("File filename.txt not found\n");
  }
```

Usually this happens in a function, so you need to return:

```
File *f = fopen("filename.txt","r");
if(f==0) {
  printf("File filename.txt not found\n");
  return -1;
}
```

Of course, there may be several reasons that you can't open the file. It may not exist, or there may be a permission error, or there may be an IO error (disk failure). In C this information is stored in a global variable called **errno** which you need to inspect:

```
File *f = fopen("filename.txt","r");
if(f==0) {
  if(errno==ENOENT) {
    printf("File filename.txt not found\n");
    return -1;
  }
  if(errno==EPERM){
     printf("You do not have permission to open filename.txt\n");
     exit(0);
  }
  if(errno==EIO){
    printf("I/O error on opening the file. Get help! Your hard drive is
crashing!\n");
    get_help();
  }
}
```

To read the file we need to use the **fread()** function, which can also return errors --- it can return end-of-file, but it can also return I/O errors and other errors as well. Then you need to close the file with **close()**, and that can generate errors too. Each time you call a system call, you need to check all of the return codes and handle them appropriate. Ick. This is getting way out of control!

Java takes a different approach: any method can *throw an exception.* This creates an error condition which must be **caught**. Exception handling roughly follows this procedure:

1. Is the exception caught in the method that generated the exception?

    1. if so, run the code that catches the exception.
    2. If not, go to the method that called the current method and repeat step #1
2. If you reach the top of the callstack, then this is an *uncaught exception.* Print the stack trace and terminate the current thread.

In your code, you were getting a stack trace and having your program terminated because you were not catching the Null reference exception. Your error, incidentally, is not that you were not catching the exception - - - your error was that you were not setting the instance variable. **In general you should not catch the null reference exception when you are developing code.** You should, instead, write your code so that you don't get null reference exceptions.

Typical Java exceptions

Here are some typical Java exceptions and examples of what an intelligent program might want to do with them:

Exception	**Typical uses**
ArithmeticException	Division by 0.
IllegalFormatConversionException	Illeagal format in a printf statement
IOException	Something bad happened when your program was trying to read or write a file. Perhaps the file didn't exist, perhaps the disk is failing.
URISyntaxException	An invalid URL or URI was given to a parser. For example, you may have tried to open the URL httpx://www.nps.edu/.
PrintException	Your program tried to print something, and for some reason it can't print.
SQLException	Your program sent invalid SQL to the database server.
UnsupportedLookAndFeelException	Your program asked to use a look and feel that is not available on the current platform. For example, you may have told your program to use the Macintosh look and feel and it may be running on Windows.
SAXException	You attempted to parse invalid XML

Catching One Exception

Here is some simple code that calculates the mysterious B function:

```
public class B {
    int total=0;
    public B(int startValue){
        this.total = startValue;
    }
    int calc(){
        /* Calculate a complicated function of i,j and k */
        for(int q = 1; q< 20;q+=1){
            total += total/q;
            total += (total-30) / (total);
        }
        return total;
    }

    public static void main(String[] args){
        int initialValue = Integer.parseInt(args[0]);
        B b = new B(initialValue);
        System.out.printf("b.calc()=%d %n",b.calc());
    }
}
```

Let's run it a few times:

```
$ java B 1
b.calc()=-20

$ java B 2
b.calc()=8
```

And here's the exception:

```
$ java B 3
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at B.calc(B.java:10)
        at B.main(B.java:19)
```

So now we have a complicated piece of code which sometimes generates a fatal error. We could try to figure out every possible condition that might generate the error, but that's not the correct approach. This is an error based on user input. We just need to tell the user that the function doesn't work with that input. To do this, we need to catch the exception:

```
    public static void main(String[] args){
        int initialValue = Integer.parseInt(args[0]);
        B b = new B(initialValue);
        try{
            System.out.printf("b.calc()=%d %n",b.calc());
        } catch (ArithmeticException e){
            System.out.printf("The B function cannot be calcualted for %d %
n",initialValue);
        }
    }
```

When this runs, we get:

```
$ java B 3
The B function cannot be calculated for 3
$
```

Catching Multiple Exceptions

It turns out that there are two different ways that the user can give us invalid input:

1. The user can supply a non-numeric argument
2. The user can supply no argument

These generate different exceptions:

```
$ java B f
Exception in thread "main" java.lang.NumberFormatException: For input string: "f"
        at java.lang.NumberFormatException.forInputString
(NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:447)
        at java.lang.Integer.parseInt(Integer.java:497)
        at B.main(B.java:16)
$ java B
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at B.main(B.java:16)
$
```

This code catches both exceptions:

```
    public static void main(String[] args){
        int initialValue;
        try {
            initialValue = Integer.parseInt(args[0]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("You must supply at least one argument");
            return;
        } catch (NumberFormatException e){
            System.out.println("You must supply a number.");
            return;
        }

        B b = new B(initialValue);
        try {
            System.out.printf("b.calc()=%d %n",b.calc());
        } catch (ArithmeticException e){
            System.out.printf("The B function cannot be calcualted for %d %
n",initialValue);
        }
    }
```

Java Exception Classes and Instances

In this code:

```
        } catch (ArithmeticException e){
```

The variable **e** is your exception variable. it's actually a local variable that is set to be the value of the exception.

There are some important things about Java exceptions that make them very different from C++ exceptions (but very similar to Python exceptions):

- Exceptions generated in your code are represented as **instances** of **Exception Classes**
- All exception classes inherit from the base class *java.lang.Exception* which, itself, is a subclass of *java.lang.Throwable*
- You can create your own exceptions by subclassing Exception.
- You can throw your own exceptions with the **throw** statement.
- There is another subclass of Throwable called Error. These generally aren't caught.

Printing the stack trace

Even if you catch the exception, you may still want to print the stack trace. You do this by calling the exception object's **printStackTrace()** method, like this:

```
        B b = new B(initialValue);
        try {
            System.out.printf("b.calc()=%d %n",b.calc());
        } catch (ArithmeticException e){
            System.out.printf("The B function cannot be calcualted for %d %
n",initialValue);
            e.printStackTrace();
        }
```

And here is the code in action:

```
$ java B 3
The B function cannot be calcualted for 3
java.lang.ArithmeticException: / by zero
        at B.calc(B.java:10)
```

```
        at B.main(B.java:30)
$ %
```

Notice that the stack trace is printed for **where the exception happened,** not where it was printed.

The **finally** statement

You may want to have some code that **always runs**:

- If an exception is thrown.
- If no exception is thrown.
- If there is a **return** statement.

This is what the **finally** statement is for. You might use it:

- To erase a temporary file.
- To print some kind of special message.

Here is an example:

```
       B b = new B(initialValue);
       try {
           System.out.printf("b.calc()=%d %n",b.calc());
       } catch (ArithmeticException e){
           System.out.printf("The B function cannot be calcualted for %d %
n",initialValue);
           e.printStackTrace();
           return;
       } finally {
           System.out.println("This code always runs");
       }
```

Notice that the **return** statement now appears after the **e.printStackTrace();**

Here's what it looks like when the code runs:

```
$ java B 2
```

```
b.calc()=8
This code always runs
$ java B 3
The B function cannot be calcualted for 3
java.lang.ArithmeticException: / by zero
        at B.calc(B.java:10)
        at B.main(B.java:30)
This code always runs
$
```

Checked vs. Unchecked Exceptions

Java programmers use the term **checked exception** to indicate that method call that might generate an exception that happens within an appropriate **try {} catch {}** block. The phrase **unchecked exception** refers to an exception that is thrown without being in an appropriate block.

Checked Exceptions

- Used for runtime error conditions that are *expected to occur* and must be handled.
- Occur even after a program has been tested and deployed in the field.
- Must be declared if you throw them (see below)

Unchecked Exceptions

- Used for runtime errors that are not expected to occur.
- Things you want to catch when the program is in development.
- Usually the result of a bug in your program.
- Represent *defects in the program*

The compiler will force you to check all methods that can throw an exception unless the exception is a subclass of **RuntimeException** or any of its subclasses.

java.lang.ArithmeticException is a subclass of **java.lang.RuntimeException** and is therefore **unchecked**. You do not need to check for this exception. If it were a checked exception, then every time you wanted to have a division, you would need to put it in a **try {} catch {}** block, and that would be a huge pain.

Throwing Exception

So let's say you want to throw an exception....

In the ChargedBot() class, you may have a method that computes the electrostatic attractive force between two bots:

```
    public double electrostaticForce(ChargedBot cb){
        double range = rangeToObject(cb);
        return -(charge * cb.charge) / (range*range);
    }
```

What happens if two bots are in the same position?

- range will be 0
- This will generate a divide-by-zero error.
- This will make NO SENSE TO THE PROGRAMMER!

We want to generate an ArithmeticException, but don't want it to say "divide by 0." We want it to say **something that will make sense to the programmer.**

Here is the code that does it:

```
    public double electrostaticForce(ChargedBot cb){
        double range = rangeToObject(cb);
        if(range==0) throw new ArithmeticException("Two bots cannot occupy the
 same space");
        return -(charge * cb.charge) / (range*range);
    }
```

Notice:

- To throw an exception, we say **throw <object>**
- Before we can throw, we need to make the exception object (**new ArithmeticException("")**)
- ArithmeticException's constructor takes an optional **String s** which creates a *detail message* . http://java.sun.com/j2se/1.5.0/docs/api/java/lang/ArithmeticException.html

Instead of saying this:

```
            if(range==0) throw new ArithmeticException("Two bots cannot occupy
 the same space");
```

We could use *ugly code*:

```
             if(range==0){
                ArithmeticException e = new ArithmeticException("Two bots
 cannot occupy the same space");
                throw e;
              }
```

That **e** is the same instance that would be picked up by the appropriate **catch** statement.

Throwing Another Exception

We could **bullet-proof** the ChaseBot.setTarget method to make sure that nobody ever sets a bot to chase itself:

```
    public void setTarget(FlatlandObject aTarget){
        if(aTarget==this) throw new RuntimeException("ChaseBots cannot
 chase themselves");
        target = aTarget;
    }
```

Making your own exceptions

Instead of having electrostaticForce() throw an ArithmeticException, we may wish to create a new Exception class --- call it a **RangeException.**

```
   public class RangeException extends RuntimeException {
       RangeException(String message){
           super(message);
```

```
        }
    }
```

Then to throw it, you would say:

```
            if(range==0) throw new RangeException("Two bots cannot occupy the
same space");
```

There are two places you could put the RangeException:

- In a file called **RangeException.java**
- As an inner class within the ChargeBot.java file, like this:

```
    /** Inner exception class for invalid electrostatic
force calculation
     */
    public class RangeException extends RuntimeException {
        RangeException(String message){
            super(message);
        }
    }

    /
**
     * returns the electrostatic force applied by the
provided object.
     * where the electrostatic force is proportional to
the product
     * of the two charge divided by the distance
between them.
     */
    public double electrostaticForce(ChargedBot cb){
        double range = rangeToObject(cb);
        if(range==0) throw new RangeException("Two bots cannot occupy the
same space");
        return -(charge * cb.charge) / (range*range);
    }
```

Reporting Exceptions

Methods that can throw a checked exceptions must report which exceptions they throw.

Because RangeException extends RuntimeException, it doesn't need to be checked.

Here is a weird X class and related function:

```
public class X {
    int tally = 0;
    public int calc(int n){
        if(n==5) throw new RuntimeException("We do not like n==5");
        tally += n;
        return tally+n*n;
    }

    public void print10(){
        for(int i=0;i<10;i++){
            System.out.printf("calc(%d)=%d %n",i,calc(i));
        }
    }

    public static void main(String[] args){
        X obj = new X();

        obj.print10();
    }
}
```

And here's it running:

```
$ java X
calc(0)=0
calc(1)=2
calc(2)=7
calc(3)=15
calc(4)=26
Exception in thread "main" java.lang.RuntimeException: We do not like n==5
        at X.calc(X.java:4)
        at X.print10(X.java:11)
```

```
        at X.main(X.java:18)
$
```

If calc threw an Exception and not a RuntimeException, it wouldn't compile.

Here is the modified code:

```
    public int calc(int n){
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }
```

And here is what happens when it is compiled:

```
$ javac X.java
X.java:4: unreported exception java.lang.Exception; must be caught or declared to
be thrown
        if(n==5) throw new Exception("We do not like n==5");
                 ^
1 error
$
```

The calc() method needs to be modified:

```
    public int calc(int n) throws Exception{
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }
```

But the X.class still won't compile:

```
$ javac X.java
X.java:11: unreported exception java.lang.Exception; must be caught or declared
```

```
to be thrown
            System.out.printf("calc(%d)=%d %n",i,calc(i));
                                                    ^
1 error
$
```

Look at the code:

```
public class X {
    int tally = 0;
    public int calc(int n) throws Exception{
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }

    public void print10(){
        for(int i=0;i<10;i++){
            System.out.printf("calc(%d)=%d %n",i,calc(i));
        }
    }

    public static void main(String[] args){
        X obj = new X();

        obj.print10();
    }
}
```

- print10 doesn't catch the Exception().
- You can see this by looking at the code.
- So can the compiler!

Two ways to get the code to compile

Approach #1 - - - Catch the Exception

```
public class X {
```

```
    int tally = 0;
    public int calc(int n) throws Exception{
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }

    public void print10(){
        for(int i=0;i<10;i++){
            try{
                System.out.printf("calc(%d)=%d %n",i,calc(i));
            } catch(Exception e){
                System.out.println("Woot! Got an exception:");
                e.printStackTrace();
            }
        }
    }

    public static void main(String[] args){
        X obj = new X();

        obj.print10();
    }
}
```

```
$ javac X.java
$ java X
calc(0)=0
calc(1)=2
calc(2)=7
calc(3)=15
calc(4)=26
Woot! Got an exception:
java.lang.Exception: We do not like n==5
        at X.calc(X.java:4)
        at X.print10(X.java:12)
        at X.main(X.java:23)
calc(6)=52
calc(7)=72
calc(8)=95
calc(9)=121
```

Approach #2 - - - Declare print10() as throwing the exception

```
public class X {
    int tally = 0;
    public int calc(int n) throws Exception{
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }

    public void print10() throws Exception {
        for(int i=0;i<10;i++){
            System.out.printf("calc(%d)=%d %n",i,calc(i));
        }
    }

    public static void main(String[] args){
        X obj = new X();
        obj.print10();
    }
}
```

Compile it:

```
$ javac X.java
X.java:17: unreported exception java.lang.Exception; must be caught or declared
to be thrown
        obj.print10();
                   ^
1 error
$
```

Now the compiler knows that main() is not catching the exception. It needs to be fixed too:

```
public class X {
    int tally = 0;
```

```
    public int calc(int n) throws Exception{
        if(n==5) throw new Exception("We do not like n==5");
        tally += n;
        return tally+n*n;
    }

    public void print10() throws Exception {
        for(int i=0;i<10;i++){
            System.out.printf("calc(%d)=%d %n",i,calc(i));
        }
    }

    public static void main(String[] args){
        X obj = new X();
        try{
            obj.print10();
        } catch(Exception e){
            System.out.println("Woot! Got an exception:");
            e.printStackTrace();
        }
    }
}
```

```
$ javac X.java
$ java X
calc(0)=0
calc(1)=2
calc(2)=7
calc(3)=15
calc(4)=26
Woot! Got an exception:
java.lang.Exception: We do not like n==5
        at X.calc(X.java:4)
        at X.print10(X.java:11)
        at X.main(X.java:18)
$
```

Note:

- This time the loop doesn't restart. (why not?)

Notes on Exceptions

- Exceptions are SLOW - - - use them only for exceptional conditions (Bloch Item 39)
- Avoid Unnecessary use of checked exceptions (Bloch Item 41) - - - Frequently the exceptions you throw should be unchecked.
- Throw exceptions appropriate to the abstraction (Bloch item 43) (NoSuchUser instead of SQLException) [1]

Assertions

Java has a system called **Assertions** which allow you to specify invariants. This is done with the **assert** keyword.

This code:

```
    public void setTarget(FlatlandObject aTarget){
        if(aTarget==this) throw new RuntimeException("ChaseBots cannot
chase themselves");
        target = aTarget;
    }
```

Could be rewritten like this:

```
    public void setTarget(FlatlandObject aTarget){
        assert aTarget!=this;
        target = aTarget;
    }
```

When you run a program with assertions, you must give the Java runtime the **-ea** or **-enableassertions** flag (enable assertions).

Here is a little demo program:

```
public class x {
    public static void main(String[] args){
        assert false;
```

```
}
```

And let's run it:

```
$ java x
$ java -ea x
Exception in thread "main" java.lang.AssertionError
        at x.main(x.java:3)
$ java -enableassertions x
Exception in thread "main" java.lang.AssertionError
        at x.main(x.java:3)
```

Assertion theory:

- Put lots of assertions in when developing the code.
- Run with -ea during Q&A.
- Run without -ea when the product is shipped and running in the field.

Advantages of assertions:

- Clear language Support
- Clearly document expected internal state (e.g., assert radius>0)
- Easy to disable in production code without having to edit your source file.

Disadvantages of assertions:

- They get disabled in the field, so people don't know if the program is not performing properly.
- Hard to give meaningful feedback other than **assertion failed.**
- An assertion statement that has a side-effect will change behavior if assertions are turned off.

BAD PROGRAM EXAMPLE:

```
assert (i=j) == k;
```

See also: Programming with Assertions

Readings

For Tuesday:

- Java Tutorial trail on exceptions

For Wednesday:

- Teaching Defensive Programming in Java, Marsha Zaidman, 2003.

This Week:

- Day 7: Exceptions, Assertions, and Threads, in *Teach Yourself Java in 21 days*

If you purchased Head First Java, you should now be familiar with these chapters:

- Chapter 1, Breaking the Surface
- Chapter 2, A Trip to Objectville
- Chapter 3, Know your Variables
- Chapter 4, How Objects Behave
- Chapter 7, Better Living in Objectville
- Chapter 9, Life and Death of an Object
- Chapter 10, Numbers Matter

For this week, read:

- Chapter 11, Risky Behavior.

Technologies

See Also

Checked vs. unchecked exceptions

- http://www.javapractices.com/topic/TopicAction.do?Id=129
- http://www.onjava.com/pub/a/onjava/2003/11/19/exceptions.html
- http://www.ibm.com/developerworks/java/library/j-jtp05254.html

- http://www.psynixis.com/blog/2007/06/06/the-ungreat-checkedunchecked-exceptions-debate/
- http://java.sun.com/docs/books/tutorial/essential/exceptions/runtime.html

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

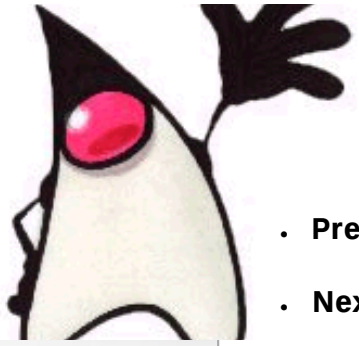
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



k 05: Basic Graphics and GUI with AWT Swing

- **Previous:** Week 04: I/O
- **Next:** Week 06: Java 2D and 3D
- **Slides:** Media:Week5.pdf Week 5 slides are posted.
- Media:Week5b.pdf Tuesday's slides
- Media:Week5c.pdf Thursday's slides

	Basic Graphics and GUI with AWT and Swing
Week	5
shortname	Swing

Contents

Goals

- Anonymous classes and pseudo-function pointers
- Swing
- HelloWorldSwing (anonymous classes)
- Basic GUI components
- Containment Hierarchy
- Frames, event-driven programming,
- Events & Listeners.
- http://java.sun.com/docs/books/tutorial/uiswing/components/jcomponent.html

Notes

Additions onto the class notes:

- The Java Tutorial has a nice page on Using Top-Level Containers JFrame, JDialog, and JApplet.

JFrame

JComponent

Layout Managers

Layout managers describe how JComponents added to a JComponent or a JFrame are layed out.

- Java Layout Page

The one "gatcha," as Sun calls it, is that the *contentPane* of the JFrame is a `Container` and not a `JComponent`. Sun says you need to do one of the following:

1. Typecast the return value.
2. Create your own component to cover the content pane.

Sun recommends the following. Here is an easy way to do it:

```
//Create a panel and add components to it.
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.setBorder(someBorder);
contentPane.add(someComponent, BorderLayout.CENTER);
contentPane.add(anotherComponent, BorderLayout.PAGE_END);


topLevelContainer.setContentPane(contentPane);
```

Fourth GUI in the SVN repository has been updated to follow this approach.

Consider a simple Swing Program:

```
import javax.swing.*;

public class FirstGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,300);
       frame.setVisible(true);
    }
}
```

This program will create a window with no functionality. All Swing objects are JFrame objects. This includes windows, buttons, etc.

Now we add a clickable button

```
public class FirstGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton button = new JButton("click me");
        frame.getContentPane().add(button);
        frame.setSize(300,300);
        frame.setVisible(true);
    }
}
```

See Also

Some good examples of Swing applications

- http://examples.oreilly.com/jswing2/code/
- http://www.javabeginner.com/java-swing-tutorial.htm

Information on CSS

- http://www.w3schools.com/css/

Readings

- Read the Wikipedia article on the Therac-25
- Please skim the Therac-25 article
- A streaming lecture on the Therac-25 [1]

- **Previous:** Week 04: I/O
- **Next:** Week 06: Java 2D and 3D

Category: JavaWeek

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

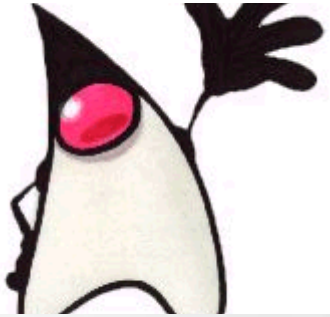
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Week 06: More Swing

(Redirected from Week 06: Java 2D and 3D)

- **Previous:** Week 05: Basic Graphics and GUI with AWT and Swing
- **Next:** Week 07: Unicode, Text & Internationalization
- Week 6 Slides - Monday
- Week 6 Slides - Thursday
- Media:Lecture15.pdf

	Java 2D and 3D
Week	6
shortname	2D and 3D

Contents

Goals

- Applets
- Really understand Swing
- Building GUI's with the NetBeans GUI builder.

Class Notes

- A mime.types file
- IANA's official mime.types

Applet Examples

- Java3D with applet tags
- Week6 Applet Demo

NetBeans GUI Builder (Matisse)

- GUI Builder home page
- Quick Start Guide
- Java GUIs and Project Matisse Learning Trail at netbeans.org
- John O'Conner's Java.net blog post

Readings

By the end of this week you should be comfortable writing programs using JFC/Swing. In our book *Teach Yourself Java in 21 Days*you should read and be familiar with the material in these chapters:

- Day 9: Working with Swing
- Day 10: Building a Swing Interface
- Day 11: Arranging Components on a User Interface

- Day 12: Responding to User Input
- Day 13: Using Color, Fonts, and Graphics

If you are working in *Head First Java,* be familiar with these chapters:

- Chapter 12: A Very Graphic Story
- Chapter 13: Work on your Swing

Wednesday Reading: Originally we were going to read this article:

- Visualizing Java in Action, Steven P. Reiss, Software Visualization, 2003

Instead, we will try something different.

Pick one of the following Java visualization toolkits. Spend 30 minutes learning what you can about it. Read the documentation, look at the screen shots, see if you can download a demo and get something to work. Be prepared to speak for 2-3 minutes about the toolkit of your choice (those in bold were from the original list):

- **JUNG**
- **Improvise**
- Graph Interface Library (GINY) - Java
- Gravisto: Graph Visualization Toolkit - An editor and toolkit for developing graph visualization algorithms.
- HyperGraph - Hyperbolic trees, in Java. Check out the home page. Try clicking on the logo...
- **InfoViz Toolkit - Java, originally developed at INRA. Does not appear to be currently maintained** Tmrashid 08:41, 13 February 2008 (PST)

- Jdigrah - Java Directed Graphs.
- JGraphT - A Java visualization kit designed to be simple and extensible.
- Linguine Maps - An open-source Java-based system for visualizing software call maps.
- **Perfuse - A Java-based toolkit for building interactive information visualization applications**
- Rox Graph Theory Framework - An open-source plug-in framework for graph theory visualization.
- TouchGraph - Library for building graph-based interfaces.
- VisAD - A Java component library for interactive and collaborative visualization.
- **The Visualization Toolkit - C++ multi-platform with interfaces available for Tcl/Tk,** Java and Python. Professional support provided by Kitware.
- Zoomable Visual Transformation Machine - Java. Originally started at Xerox Research Europe.
- **Processing.org - A new language for doing graphics and visualization.**

References

- javax.swing.JPanel
- java.awt.Desktop
- java.awt.SystemTray
- Article about using the System Tray
- Painting in AWT and Swing: Good Painting Code Is the Key to App Performance, by Amy Fowler

- **Previous:** Week 05: Basic Graphics and GUI with AWT and Swing
- **Next:** Week 07: Unicode, Text & Internationalization

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

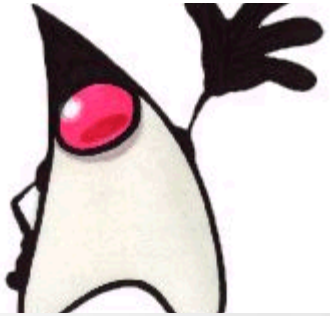
Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file

- Special pages
- Printable version
- Permanent link



Week 07: Unicode, Text & Internationalization

- **Previous:** Week 06: Java 2D and 3D
- **Next:** Week 08: Threads and the System

	Unicode, Text & Internationalization
Week	7
shortname	Text

- Media:week7.pdf

Contents

Goals

- Understand ASCII, Latin1 and Unicode
- Learn about Lucene, the open source indexing kit.
- Build an application using Lucene that will index the files on your computer.

Lucene

Lucene is the open source indexing kit. Lucene can build an index of documents on your hard drive or on a website, and then provide you with tools for rapidly searching that index for words or phrases.

We are learning about Lucene for several reasons:

- Working with text is hard, and Lucene does an excellent job.
- In many of your programs you may wish to rapidly search through some data. Lucene is a fast way to do it.
- You're almost always better off using debugged code than trying to develop something on your own.
- Using Lucene will teach you more about using other people's code.

Below are some links that you may find helpful in trying to understand Lucene:

- Lucene home page
- JavaDocs for Lucene 2.3
- Lucene getting started guide
- Lucene Download
  - Lucene command-line demo (try it!)
- org.apache.lucene.demo.SearchFiles source
- org.apache.lucene.demo.IndexFiles source

See Also

- WordNet

Readings

We were going to read this:

- Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences, Lutz Prechelt, October 1999

You are still welcome to read it, but we will be discussing this in class:

- Your Passport to Proper Internationalization, Benson I. Margulies, Dr. Dobb's Journal, May 2000.

Please read Chapter 2 of the Unicode book:

- Chapter 02

- **Previous:** Week 06: Java 2D and 3D
- **Next:** Week 08: Threads and the System

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments

Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

eek 08: Threads and the System

- **Previous:** Week 07: Unicode, Text & Internationalization
- **Next:** Week 09: Network programming with Java

	Threads and the System
Week	8
shortname	Threads

Contents

Goals

- Threads
- Synchronization
- Serialization

Slides

- Media:week8.pdf
- The Life Cycle of a Thread

Notes on Class Topics

Serialization

When an object is serialized, java writes a '*serialVersionUID* into the object archive. This is supposed to be a unique version number for the object that you are using. Whenever you change the object's instance variables that are not transient, you need to change the serialVersionUID.

If you do not declare your own **serialVersionUID,** the compiler will figure one up for you at runtime. The problem with this approach, though, is that newer versions of your object can't read older versions.

Some people put things like this into their class files:

//Added to make warning go away. private static final long serialVersionUID = 1L;

This is usually a bad idea, because it means that any change to the class will not result in the object serialVesionUID being saved in the archive. That's fine if you want to write code that can handle reading the older versions of your objects.

To get the UID, you can use the java program **serialver**.

Here is a sample class:

```
import java.io.*;

public class y implements Serializable {
    int f;
}
```

To get the serialVersionUID, compile it and then run **serialver**:

```
$ javac y.java
$ serialver y
y:    static final long serialVersionUID = 1534365507996228659L;
$
```

The class should now look like this:

```
import java.io.*;

public class y implements Serializable {
    int f;
    static final long serialVersionUID = 1534365507996228659L;
}
```

If you make a change to the class, you get a different **serialVersionUID**:

```
import java.io.*;

public class y implements Serializable {
    int f;
    int g;
}
```

```
$ javac y.java
$ serialver y
y:    static final long serialVersionUID = -4791259395666985996L;
$
```

Note that Eclipse (but not NetBeans) can figure this out for you if you click the lightbulb.

- Sun documentation on **serialver** command
- Sun documentation on Serializable
- Discussion on the Sun chat boards

Reflection

See Also

- Managing Volatility: Guidelines for using volatile variables, Brian Goetz, Senior Staff Engineer, Sun Microsystems, IBM Developer Works.
- Multithreaded toolkits: A failed dream?, Graham Hamilton, java.net developer's blog, October 19, 2004.
- Java Performance Chapter 4 - I/O Performance - Nice discussion about Basic I/O,

Buffered Streams, and Serialization.

Readings

Writing Solaris Device Drivers in Java

See Also

- http://en.wikipedia.org/wiki/Trampoline_(computers)

- **Previous:** Week 07: Unicode, Text & Internationalization
- **Next:** Week 09: Network programming with Java

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Week 09: Network programming with Java

- **Previous:** Week 08: Threads and the System
- **Next:** Week 10: Style and Performance

	Network programming with Java
Week	9
shortname	net

Contents

Goals

You should know this by the end of this week:

- Theory: TCP/IP, client/server computing, OSI Stack, Sockets
- SOAP, XMLRPC, and REST
- Downloading a web pages

You may wish to know this:

- JavaSocket, Server Socket

Lesson Plan: Monday

- Review Final Project
- Show how to add a web page to the wiki
- Discuss Networking (See Goals above)

Lesson Plan: Tuesday

Things to think about:

- We use the HttpURLConnection, InputStreamReader, and BufferedReader classes to read from a URL.
  - Is it better to read the entire web page at one time, or to read it line-by-line, or something else?
  - Can you always read an **entire** web page?

Lesson Plan: Wednesday

- Go over modified single-threaded server
  - Show with one client
  - Show with two simultaneous clients
- Go over modified multi-threaded server
  - Examine synchronize methods.
  - Show with one client

- Show with two simultaneous clients

Readings

Please review the following chapters in Teach Yourself Java 6 in 21 Days:

- Day 17, Communicating Across the Internet

You may also find these chapter sot be useful for work that we have previously discussed:

- Day 14, Developing Swing Applications (especially the section about java Web Start)
- Day 16, Serializing and Examining Objects (just read pages 433-442)
- Day 15, Working with Input and Output (especially pages 422-429)

There is no reading for Wednesday; instead, David Pollack will give a special presentation about Scala Programming Language on Thursday.

Good article on threading:

- Threading Lightly, Part 1: Synchronization is not the enemy

---

- **Previous:** Week 08: Threads and the System
- **Next:** Week 10: Style and Performance

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

e and Performance

- **Previous:** Week 09: Network programming with Java

	Style and Performance
Week	10
shortname	style

Possible Topics:

- Reflection
- Memory Management
- Profiling
- GCJ
- Security Policy
- Java RMI

Contents

Reflection

Here is a little program that prints the methods for java.lang.String:

```
import java.lang.reflect.*;

public class ReflectionDemo {
    public static void main(String[] args){
        System.out.println("Get a String class and show the methods that it
implements");

        try {
            Class stringClass = Class.forName("java.lang.String");
            System.out.println("Class: "+stringClass.getName());
            System.out.println("Methods:");
            Method[] methods = stringClass.getMethods();
            for(Method m: methods){
                System.out.println("   "+m.getName());
            }
        } catch (ClassNotFoundException e){
            System.out.println("Class not found: "+e.getMessage());
        }
    }
}
```

Output 1

Change the print line to do this see the return type of each:

```
                System.out.printf("  (%s) %s %n",m.getReturnType().getName(),m.
getName());
```

Output 2

Notes For Effective Java

Comparable

- Interface Comparable<t>
- article on pre-Java5 Comparable

Advantages of new Comparable:

- You don't need to do instanceof (type safety)
- Less code to implement.

Here is a simple class that implements comparable:

```
public class Person implements Comparable<Person>{
    String name;
    int age;
    public int compareTo(Person p){
        int r = name.compareTo(p.name);
        if(r!=0) return r;
        if(age<p.age) return -1;
        if(age>p.age) return 1;
        return 0;
    }
}
```

Clone

- Note that "cloneable" is misspelled.
- Please see the CloneDemo.java and Person.java in the subversion repository.
- If the class you are extending implements clone(), your class must implement clone()
- To clone:
  - First make a copy with super.clone()
  - Next, clone (or copy) the instance variables that are mutable.
- JavaPractices.com recommends avoiding clone entirely:
  - Avoid clone
  - use Copy constructors instead.
  - Use Immutable objects when possible.

String is an immutable object. They have a lot of advantages:

- Are thread-safe
- Do not need a copy constructor
- Do not need an implementation of clone (since copying the fields is fine)
- Lots of other advantages (see Immutable objects).

Choices for Text Field Validation

Handling Document Events

The correct way to do validation of a text field is by creating a DocumentEvent handler class and object. The object receives document changed events. a removeUpdate() event is generated when text is removed from the text field and an insertUpdate() event is generated when text is added.

Here is some code from the JPanel constructor of my test program:

```
        initComponents();
        textField.getDocument().addDocumentListener(new DocumentListener() {
            public void changedUpdate(DocumentEvent e){
                String text = textField.getText();
                dlStatus.setText("changedUpdate: textField has " + text.length() + "
characters");
            }
            public void removeUpdate(DocumentEvent e){
                String text = textField.getText();
                dlStatus.setText("removeUpdate: textField has " + text.length() + "
characters");

            }
            public void insertUpdate(DocumentEvent e){
                String text = textField.getText();
                dlStatus.setText("insertUpdate textField has " + text.length() + "
characters");
            }
        });
        dlStatus.setText("dlListener created");
```

the **initComponents()** is put in by the NetBeans GUI builder. This creates an anonymous subclass of the DocumentListener interface. The changedUpdate() event is called when a document's ATTRIBUTES change (like bold), not when its text changes.

This is a better approach for input validation than picking up the keyTyped() events because it will pick up changes to the "document" (the text of the text field) that are caused by copy and paste.

Another way to do it

Another way to make this work is to use java's **invokeLater** method which causes a piece of code to be run *after* the current event is processed. This works (as shown below), but it has a problem: document change events that are not the result of keyboard characters (like a cut and paste) will not cause the textFieldKeyTyped event to be sent.

```
    private void textFieldKeyTyped(java.awt.event.KeyEvent evt) {

        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
                String text = textField.getText();
                statusLabel.setText("textField has " + text.length() + "
 characters");

            }
        });

    }
```

References

Lots of people seem confused by this. Here are some relevant URLs:

- JTextField documentation.
- Java Forum Key Pressed Event Problem discussion
- Java Tutorial How to write a document listener

Readings

- The Reflection API

Readings From *Effective Java* :

- Chapter 1 (bits) - on Google Books
- Chapter 3 - Methods Common to All Objects
- Chapter 5 - Substitutes for C Constructions
- Chapter 6 - Methods
- Chapter 7 - General Programming

- **Previous:** Week 09: Network programming with Java

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

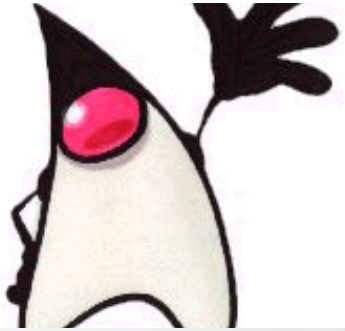
- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link



Week 11: The Big Finish

	The Big Finish
Week	11
shortname	Finish

Contents

Goals

- Answer all outstanding questions
- Final Project Presentations
- Tuesday: SQL & JDBC
- Wednesday: JAVA Security
- Thursday: My favorite Tools

Bloch, Chapter 4: Classes and Interfaces

Item 12: Minimize the accessibility of classes and members

find copyright violations

- Make each class or member as inaccessible as possible.
- Accessibility hiearchies:
  - public
  - protected
  - package-private (default access)
  - Private
  - local
- Make things final unless they need to be modifiable
- Make things static unless they need to access instance variables.
- Don't use public fields---if a field is non-final or a final reference to a mutable object, you give up the ability to:
  - validate a field

- Take an action when a field is modified
- Contents of a final array are modifiable!

Item 13: Favor Immutability

1. Don't provide any methods that modify the object (mutators)
2. Ensure that no methods may be overridden
3. Make all fields final
4. Make all fields private
5. Ensure exclusive access to any mutable components

Advantages of immutable objects:

- They are thread-safe and require no synchronization
- They can be shared freely
- The internals can also be shared
- Immutable objects make great building blocks for other objects
- Use static factories instead of constructors

Item 14: Favor composition over inheritance

- Subclasses need to understand their parent classes

Item 15: Design and Document for inheritance or else prohibit it

Item 16: Prefer interfaces to abstract classes

- Existing classes can be easily retrofitted to implement new interfaces
- Interfaces are ideal for defining mixins.

Item 17: Use interfaces only to define types

- Don't define constants in interfaces

Item 18: Favor static member classes over nonstatic

Sign-up for Final Projects

Thursday

0800 - Craig Schwetje

Friday

0800 - 0815: Jason Nelson

0815 - 0830: Brian Hawkins

0830 - 0845: Tariq Rashid

0845 - 0900:

0900 - 0915:

0915 - 0930:

0930 - 0945:

0945 - 1000:

Category: JavaWeek

CS 3773

- Main Page
- Week-by-week Outline
- Assignments
- Readings

Wiki

- Recent Changes

Search

Go Search

Toolbox

- What links here
- Related changes
- Upload file
- Special pages
- Printable version

- Permanent link