CS61BL

Lecture 1:
Welcome to CS61BL!
Intro to Java and OOP
Testing
Error-handling
About me

• Name: Edwin Liao
• Email: edliao@berkeley.edu
• Office hours:
  – Thursday 3pm - 5pm
  – Friday 11am - 1pm
  – 611 Soda
  – Or by appointment
Course Info

• Website: http://inst.eecs.berkeley.edu/~cs61bl/su14/
• Piazza: https://piazza.com/berkeley/summer2014/cs61bl
• Email: cs61bl@inst.eecs.berkeley.edu
Readings

• Head First Java
• Professor Shewchuk’s lecture notes
• Concise Introduction to Data Structures in Java
Course Format

<table>
<thead>
<tr>
<th></th>
<th>CS61B</th>
<th>CS61BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture</td>
<td>🟢</td>
<td>🟣</td>
</tr>
<tr>
<td>Discussion</td>
<td>🟢</td>
<td>🟣</td>
</tr>
<tr>
<td>Labs</td>
<td>🟢</td>
<td>🟣</td>
</tr>
</tbody>
</table>
What about lectures?

• My personal philosophy:
  – Lectures = preview of what’s to come
  – Connect all topics together; the big picture
  – Run demos to teach material that would otherwise be difficult to explain
Collaboration

• Pair programming in labs
  – One partner at keyboard ("driver")
  – One partner observing ("navigator")
  – Decrease bugs!

• Partners / groups for projects
  – Groups of 2-3 for project 1
  – Groups of 3-4 for projects 2 and 3
  – Group evals turned in with projects
<table>
<thead>
<tr>
<th>Category</th>
<th>Points</th>
<th>% of Total Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project 1</td>
<td>15</td>
<td>5%</td>
</tr>
<tr>
<td>Project 2</td>
<td>30</td>
<td>10%</td>
</tr>
<tr>
<td>Project 3</td>
<td>30</td>
<td>10%</td>
</tr>
<tr>
<td>Midterm 1</td>
<td>45</td>
<td>15%</td>
</tr>
<tr>
<td>Midterm 2</td>
<td>45</td>
<td>15%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>75</td>
<td>25%</td>
</tr>
<tr>
<td>Quizzes (capped at 70%)</td>
<td>30</td>
<td>10%</td>
</tr>
<tr>
<td>Homework Assignments (capped at 90%)</td>
<td>30</td>
<td>10%</td>
</tr>
<tr>
<td>Total:</td>
<td>300</td>
<td>100%</td>
</tr>
</tbody>
</table>
Capped at ___%?

• Quizzes: 70%
• Homework assignments: 90%
• Example 1: Received 92% of all homework points
  – Capped: 92/90 -> 90/90
  – Scaled: 90/90 -> 100/100
• Example 2: Received 60% of all homework points
  – Capped: 60/90 -> 60/90
  – Scaled: 60/90 -> 67/100
• Why?
  – Murphy’s law
Cheating

- Credit sources
- Don’t copy / share code
- When in doubt, ask
Primitives

• Two types of data: primitives and references
• Examples of primitives: boolean, byte, int
• Fixed amount of memory
  – byte: 8 bits
  – int, float: 32 bits
  – long, double: 64 bits
Some OOP Lingo

- Reference: points a variable to an object
- Class: blueprints for objects; outline of what objects can do
- Method: a function associated with a particular class
Anatomy of a Class

```java
class Coin {
    double p; // Probability that this coin will show heads after a flip

    public Coin() {
        this.p = 0.5;
    }

    /**
     * Simulates a coin flip with this coin.
     * Returns true if the flip resulted in a heads.
     */
    public boolean flip() {
        return (Math.random() < p);
    }
}
```
Anatomy of a Class

```java
public class Coin {
    public double p; // Probability that this coin will // show heads after a flip

    public Coin() {
        this.p = 0.5;
    }

    /**
     * Simulates a coin flip with this coin.
     * Returns true if the flip resulted in a heads.
     */
    public boolean flip() {
        return (Math.random() < p);
    }
}
```
public class Coin {
    public double p; // Probability that this coin will
                    // show heads after a flip
    public Coin() {
        this.p = 0.5;
    }
    /**
     * Simulates a coin flip with this coin.
     * Returns true if the flip resulted in a heads.
     */
    public boolean flip() {
        return (Math.random() < p);
    }
}
public class Coin {
    public double p; // Probability that this coin will // show heads after a flip

    public Coin() {
        this.p = 0.5;
    }

    /**
     * Simulates a coin flip with this coin.
     * Returns true if the flip resulted in a heads.
     */
    public boolean flip() {
        return (Math.random() < p);
    }
}
public class Coin {
    public double p; // Probability that this coin will show heads after a flip
    public Coin() {
        this.p = 0.5;
    }
    /**
     * Simulates a coin flip with this coin.
     * Returns true if the flip resulted in a heads.
     */
    public boolean flip() {
        return (Math.random() < p);
    }
}
The `main` method

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

- Where a program begins its execution
- All programs need one to run
- Signature has to be exact!
public class Coin {

    public double p;

    public Coin() {
        this.p = 0.5;
    }

    public boolean flip() {
        return (Math.random() < p);
    }

    public static void main(String[] args) {
        Coin myCoin = new Coin();
        boolean b = myCoin.flip();
    }
}
public class Coin {

    public double p;

    public Coin() {
        this.p = 0.5;
    }

    public boolean flip() {
        return (Math.random() < p);
    }

    public static void main(String[] args) {
        Coin myCoin = new Coin();
        boolean b = myCoin.flip();
    }
}
public class Coin {

    private double p;

    public Coin(double probHeads) {
        this.p = probHeads;
    }

    public boolean flip() {
        return (Math.random() < p);
    }

    public static void main(String[] args) {
        Coin myCoin = new Coin(0.7);
        System.out.println(myCoin.flip());
    }
}
public class Coin {
    private double p;

    public Coin(double probHeads) {
        this.p = probHeads;
    }

    public boolean flip() {
        return (Math.random() < p);
    }

    public static void main(String[] args) {
        Coin myCoin = new Coin(0.7);
        System.out.println(myCoin.flip());
    }
}
Break Time!

Up next:

Testing!
Why Test?

• Option 1: Code, submit, hope it works
  – Good luck

• Option 2: Code everything, try to run it, then go back and fix bugs
  – Time consuming

• Option 3: Test your code
  – Yes!
Get a Job!

• Some companies: test your own code
  – If your code breaks stuff...
  – :(  
  – So you better get it right

• Or hire software engineers to test
  – Google: Software Engineer in Test
  – Apple: Software Developer in Test
  – Microsoft: Software Development Engineer in Test
Testing Terminology

• Unit testing
  – Writing tests for small modules of code (e.g. an individual method)
  – Rely minimally on other code

• Integration testing
  – Testing that code modules work together as intended
Testing Terminology

• Regression testing
  – Making sure that new code doesn’t introduce bugs in old code

• Stress testing
  – Go big
  – Probably won’t do too much of this in CS61BL
Test-driven Development

1. Write some test code
2. Make sure your test code fails
3. Write some production code
4. Make sure your test code passes
   – Or fix your code until it does
5. Repeat
How to Test

• It’s okay if you don’t test all cases!
  – Example: a method that takes in an `int` as input: not practical to test that method with every `int`

• But make sure you check boundary cases
  – If the input is any non-negative integer, try 0, 1, and some other positive integer(s)

• And try to test all branches
  – There’s an if-else in your code: test both branches
public class Coin {

    private double p;

    public Coin(double probHeads) {
        this.p = probHeads;
    }

    public boolean flip() {
        return (Math.random() < p);
    }

    public static void main(String[] args) {
        Coin myCoin = new Coin(0.7);
        System.out.println(myCoin.flip());
    }
}
import java.util.Scanner;

public class Coin {
    private double p;

    public Coin() {
        Scanner inputReader = new Scanner(System.in);
        double inputProb = inputReader.nextDouble();
        this.p = inputProb;
    }

    // Rest of code omitted
public Coin() {
    Scanner inputReader = new Scanner(System.in);
    double inputProb = inputReader.nextDouble();
    this.p = inputProb;
}

public Coin() {
    Scanner inputReader = new Scanner(System.in);
    try {
        double inputProb = inputReader.nextDouble();
        this.p = inputProb;
    } catch (InputMismatchException e) {
        System.err.println("You must enter a number!");
        System.exit(1);
    }
}
public Coin() {
    Scanner inputReader = new Scanner(System.in);
    try {
        double inputProb = inputReader.nextDouble();
        if (inputProb < 0 || inputProb > 1) {
            throw new IllegalArgumentException("Input not between 0 and 1!");
        }
        this.p = inputProb;
    } catch (IllegalArgumentException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    } catch (InputMismatchException e) {
    // Rest of code below
Importing Packages

import java.util.Scanner;

• Many useful packages that come with Java
  – “Java standard libraries”
  – e.g. java.util, java.io, java.swing, java.math
• You can also import your own packages
Using libraries
Vs.
Coding it yourself
Text editors + command line
Vs.
IDEs
See you next week.