Welcome to Comp215: Introduction to Program Design

Dan S. Wallach and Mack Joyner, Rice University
Welcome, welcome!

Java (really Java8)

Recursion and recursive data structures

Software engineering and design (performance, extensibility)

Some projects to stretch your skills
You may have learned Java before

In this class we’ll be teaching Java8
Biggest change to the Java language in a decade
We’ll be teaching things the “new” way, whenever possible

We’ll be building most things ourselves
We won’t be using the “collections” classes in java.util

If you’ve never written a line of Java beforehand
Work your way through one of the introductory Java MOOCs this week.
  · We emailed everybody about this at the start of the summer.

Don’t panic: We’ll move fast, but you’ll be able to keep up.
Preliminaries

Fill out our online form
http://goo.gl/forms/ArYKWBc0zY

Don’t panic: everything (including these slides) are online
Slides, syllabus, schedule: comp215.blogs.rice.edu
Weekly labs and projects, discussions, etc.: private Piazza group
Grades: Canvas (we’re not using any of the other Canvas features)
Course syllabus

These slides are just an overview. Read everything here:
http://comp215.blogs.rice.edu/course-syllabus/
Grading

**Weekly programming assignments: 50%**
Some assignments span multiple weeks. You’ll be graded each week. Specific grading criteria will vary from week to week.

**Weekly labs: 20%**
Each of these will take about an hour and is either checked-off or not. Some of these will be “review” for midterm and final.

**Midterm and final exams: 30%**
Covering topics discussed in lecture.

**Final grades will follow the usual mapping.**
80-83 is a B-, 84-86 is a B, 87-89 is a B+, 90-93 is an A-, etc.
We might curve up. We won’t curve down.
*(Last year, we curved up by almost a full grade point.)*
Grading

Weekly programming assignments: 50%
Some assignments span multiple weeks. You'll be graded each week. Specific grading criteria will vary from week to week.

Weekly labs: 20%
Each of these will take about an hour and is either checked-off or not. Some of these will be “review” for midterm and final.

Midterm and final exams: 30%
Covering topics discussed in lecture.

Final grades will follow the usual mapping.
80-83 is a B-, 84-86 is a B, 87-89 is a B+, 90-93 is an A-, etc.
We might curve up. We won’t curve down.
(Last year, we curved up by almost a full grade point.)

No extra credit. But sometimes “cool points.”
Grade disputes

Grades are posted on Canvas
Log in and see your grades any time

You have one week if you want to protest your grade
Grade disputes must be submitted in writing (email)
Include all the code, etc. we need to adjudicate the dispute
  • If we have to go look at your code online, you will be denied
You may want to speak to Dan or Mack beforehand if you’re unsure
Weekly schedule

Lectures MWF 9:00-9:50am (Sewall 301)

Labs:
Tu 4-5pm (Humanities 117)
Tu 6-7pm (Brockman 101)
W 4-5pm (Space Science 106)

Please attend the lab where you’re pre-registered, but you can go to another

• You **must** attend one of the labs each week

Any labbie can check you off afterward.

• Final deadline for lab checkoff: Friday night

**Assignments:** Out Monday morning, due Sunday 11:59pm

**Office hours:** Posted on Piazza
Late-work policy

Assignments roll out every week. Late work is heavily penalized.
0-24 hours late: Subtract 3/10 points. (Grades won’t go negative.)
24-48 hours late: Subtract 6/10 points.
>48 hours: No credit.
Special circumstances? Discuss with Dan or Mack in advance.

Assignments build on one another.
After each due date, we’ll give you reference code so you keep up.
Our advice: submit what you’ve got on time, then move on.
In some weeks, we’ll have multiple deadlines
E.g., test cases due Thursday night, code due Sunday night

Thursday deadline points are all-or-nothing (no late work accepted)
The mechanics of submission

Everything goes through Subversion (svnserve.rice.edu)
You’ll “check out” reference code.
You’ll “commit” your solution.

IntelliJ will offer to put new files in version control for you. Say yes.
Any “red” .java files are not being saved. Right-click and “add” them.
We don’t need your .class files.

Commit your files all the time.
Assume your computer will explode in a ball of fire at any time.
Version control systems are bacon saving devices!
And nothing but Subversion

Emailed zip files, printed paper, etc. are not accepted. Ever.

How to avoid having a bad Sunday evening:
Check out the project ASAP when it’s announced.
Make some trivial change and commit it immediately.
If something isn’t working, get help during the week.

If you’re exercising the late policy, you must declare this before the deadline in your README.txt file.
Primary purpose: tell your graders if you’re exercising the late policy.
Many secondary purposes (vary from week to week)

Surveys as well (optional)
We’ll ask you how much time you spent on each project, etc.
Help us improve Comp215!
Honor code issues

**Formal written policy in the course syllabus.**
You may not copy code from other students on projects or exams.
We do encourage discussion around labs.

**We use sophisticated plagiarism detection code.**
We will catch you!

**Past students have lied about Comp215 to the Honor Council.**
We’ll be posting regular reminders on Piazza.
We take this very seriously. So should you.
Getting your computer ready

You’ll need a “real” computer running Linux, Windows, or Mac OS X
Sorry, your tablet or Chromebook won’t do the job.

You’ll need to install several things on your computer
IntelliJ IDEA 2016.2 CE (or later): the free “community edition”
https://www.jetbrains.com/idea/download/
Java8 SE 8u102 (or later) JDK (not just the JRE)

Fonts: You may enjoy the “Hack” monospaced font, tuned for code readability.
Set IntelliJ to use Java8

Lab 1 walks you through this whole process with lots of screenshots. We’ll post it on Piazza.

- Fill out the web form or you won’t have access to Piazza!
Java8 in a nutshell

We’ll eventually get into lots of cool Java and IntelliJ features
Recursion, recursive data types, recursive parsing
Lambdas and first-class functions
Inheritance and subtyping
Generics, Optionals, and monadic coding styles
Unit tests, “mocks”, coverage tests, performance tests
Writing web server applications

So, why Java? Why not some other language?
Java forces you to declare all of your types in advance. Python doesn’t.

Java will catch many mistakes at compile time. Python won’t fail until runtime. But Java is way, way more verbose than Python.

• https://pythonconquerstheuniverse.wordpress.com/2009/10/03/python-java-a-side-by-side-comparison/
• http://www.programcreek.com/2012/09/java-vs-python-data-types/

Java8 makes things much nicer than older versions of Java

Benefits of Java:

Bugs caught in advance: Java lets you set rules and it enforces them.

Widely used: Web servers, Android phones.

Multi-paradigm: Object-oriented, polymorphic, multi-threaded, functional (Java8).

Fast: Sophisticated optimizing compilers, efficient memory use.

Tools: Code analysis, profilers, debuggers, development environments.

Influential: Once you know Java, it’s easy to learn C#, Scala, Swift, and more.
Java is type safe. There is no Java program that can ever:
- write beyond the end of an array
- confuse a pointer for anything else
- confuse newly allocated memory with old memory that’s still in use

In practice, Java programs fail more predictably, so they’re easier to debug.

Bonus: Java programs are (largely) invulnerable to important security attacks (buffer overflows), which are a big problem for C.

But C is much closer to the bare metal.
Used when performance is essential or resources are exceptionally limited.
And you can write hybrid C / Java programs, if you really want.
In Java, everything happens inside “classes”. Every “object” is an “instance” of a class, and you can then call “methods” on that object.

```java
public class Hello {
    private String string;

    public Hello(String string) {
        this.string = string;
    }

    public void scream() {
        System.out.println(string + "!");
    }
}
```
In Java, everything happens inside “classes”. Every “object” is an “instance” of a class, and you can then call “methods” on that object.

```java
public class Hello {
    private String string;

    public Hello(String string) {
        this.string = string;
    }

    public void scream() {
        System.out.println(string + "!");
    }
}
```

**Constructor**: takes an argument of type String, then stores it in the member variable of the same name. `this` is a pointer to the newly allocated object.
Hello, Java

In Java, everything happens inside “classes”. Every “object” is an “instance” of a class, and you can then call “methods” on that object.

```java
public class Hello {
    private String string;

    public Hello(String string) {
        this.string = string;
    }

    public void scream() {
        System.out.println(string + "!");
    }
}
```

**Method**: Anybody with a pointer to a **Hello** object can call the **scream** method. Inside the method, you can see member variables like **string**.
Hello, Java

In Java, everything happens inside “classes”. Every “object” is an “instance” of a class, and you can then call “methods” on that object.

```java
public class Hello {
    private String string;

    public Hello(String string) {
        this.string = string;
    }

    public void scream() {
        System.out.println(string + "!");
    }
}
```

**System functions**: Java has a rich standard library full of useful things. This one prints strings.
Hello, Java

In Java, everything happens inside "classes". Every "object" is an "instance" of a class, and you can then call "methods" on that object.

```java
public class Hello {
    private String string;

    public Hello(String string) {
        this.string = string;
    }

    public void scream() {
        System.out.println(string + "!");
    }
}
```

Typical usage:

Hello hello = new Hello("Rice"); // creates a Hello object
hello.scream(); // prints "Rice!"
Packages group together related functionality
edu.rice.list: classes related to lists of things
edu.rice.tree: classes related to trees of things
edu.rice.web: classes related to web servers

Classes are stored in a file with the package and class name:
package edu.rice;

public class Hello { ... }

class Hello in package edu.rice goes in edu/rice/Hello.java

IntelliJ does most of this for you automatically

- Standard style: class names are Capitalized, method names and members are lowerCaseWithBouncyCapitals
Visibility & scoping

**public**: visible to the whole world

**private**: only visible to callers within the class

**protected**: like private, but also visible to subclasses
  
  • We won’t be using this

““ = **package scope**: only visible to callers within the package

**Why?** Visibility mechanisms build a wall between *interface* and *implementation*. That’s fundamental to *enforcing abstractions*. 
We can have radically different algorithms that implement exactly the same interface.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implementation 1</th>
<th>Implementation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Set&quot;</td>
<td>Bit vector</td>
<td>Balanced tree</td>
</tr>
<tr>
<td>&quot;Queue&quot;</td>
<td>Doubly linked list</td>
<td>Balanced tree</td>
</tr>
<tr>
<td>&quot;Map&quot;</td>
<td>Hash table</td>
<td>Balanced tree</td>
</tr>
</tbody>
</table>
Java interfaces

Interfaces go in `.java` files, look like classes with only method signatures, no code (yet!).

```java
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
    boolean contains(String string);
    StringSet add(String string);
}

class StringTree implements StringSet {
    public StringSet union(StringSet other) {
        ...
    }
```

Warning: this is different from `java.util.Set`. More coming soon.
Java interfaces

Java interfaces are defined in `.java` files and look like classes with only method signatures, no code (yet!).

```java
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
    boolean contains(String string);
    StringSet add(String string);
}

class StringTree implements StringSet {
    public StringSet union(StringSet other) {
        ...
    }
}
```

Note that `StringTree` (the concrete implementation) is returning `StringSet` (the interface requirement). Most code, except constructors, will be written for the interface.

Warning: this is different from `java.util.Set`. More coming soon.
Java Interfaces

Interfaces go in \texttt{.java} files, look like classes with only method signatures, no code (yet!).

\begin{verbatim}
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
    boolean contains(String string);
    StringSet add(String string);
}

class StringTree implements StringSet {
    public StringSet union(StringSet other) {
        ...
    }
}
\end{verbatim}

// old-school usage (don't do this!)
StringSet set = new StringTree();
set.add("Hello");
set.add("Owls");

// new-school "pipelined" style
StringSet set = new StringTree().add("Hello").add("Owls");

Warning: this is different from \texttt{java.util.Set}. More coming soon.
Sometimes you find yourself writing the same thing, over and over
Software engineering rule #1: *don’t repeat yourself!* (DRY)

Every programming language has different DRY features

In Java, you might use...
- Class inheritance (i.e., traditional object-oriented coding)
- Interface default methods (new in Java8)
- Generics (more on this later)
- Static methods (i.e., traditional boring subroutines)

In Comp215, we’ll do a little of each
New feature in Java8: any interface can have “default” methods in it

```java
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
    default boolean empty() {
        return cardinality() == 0;
    }
    boolean contains(String string);
    StringSet add(String string);
}
```

If `cardinality()` is $O(1)$, you’re done.
If it’s $O(n)$, a class might instead “override” it and do something else.
New feature in Java8: any interface can have “default” methods in it

```java
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
    default boolean empty() {
        return cardinality() == 0;
    }
    boolean contains(String string);
    StringSet add(String string);
}
```

If `cardinality()` is $O(1)$, you’re done.
If it’s $O(n)$, a class might instead “override” it and do something else.

A class can implement multiple interfaces, getting possibly multiple `default` methods that overlap. But because you only get methods, not members, you avoid many ugly problems.

More later on “diamond dependencies.”
Static methods are really just functions that are hanging out with a class. You don't need to get an instance of the class.

**Static members are really global variables.**
We'll make them `final` — useful for maintaining constants.

**Example: use edu.rice.util.Log’s static methods to print debugging messages.**

```java
class LogTest {
    private static final String TAG = "LogTest";

    @Test
    public void basicTest() throws Exception {
        Log.i(TAG, "starting basicTest");
        Log.e(TAG, "some error");
    }
}
```

Static methods and members

Static methods are really just functions that are hanging out with a class. You don’t need to get an instance of the class.

**Static members are really global variables.**
We’ll make them `final` — useful for maintaining constants.

**Example:** use `edu.rice.util.Log`’s static methods to print debugging messages.

```java
package edu.rice.util;

public class LogTest {
  private static final String TAG = "LogTest";

  @Test
  public void basicTest() throws Exception {
    Log.i(TAG, "starting basicTest");
    Log.e(TAG, "some error");
  }
}
```

This is an example of a JUnit4 unit test. For most labs and projects, we’ll provide you with tests. You’ll also write your own. Lots more on unit testing coming very soon.
Static methods and members

Static methods are really just functions that are hanging out with a class. You don't need to get an instance of the class. Static members are really global variables. We'll make them final — useful for maintaining constants.

Example: use edu.rice.util.Log's static methods to print debugging messages.

```java
package edu.rice.util;

public class LogTest {
    private static final String TAG = "LogTest";

    @Test
    public void basicTest() throws Exception {
        Log.i(TAG, "starting basicTest");
        Log.e(TAG, "some error");
    }
}
```

This is much more useful to you than `System.out.println()`.

2016-03-05 14:13:49,745 [main] INFO LogTest - starting basicTest
For the most part, we won’t be using class inheritance in Comp215. In particular, we’ll never use “abstract methods”

- We’ll use Java8 interface default methods, instead

However, every Java class extends `java.lang.Object`

Some of these methods are widely used, especially:

- `String toString()`
- `boolean equals(Object o)`
- `int hashCode()`

Java won’t let you implement these from interface default methods
Simple class example: a Pair of A and B

class Pair {
    private A a;
    private B b;

    public Pair(A a, B b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || !(o instanceof Pair)) return false;

        Pair pair = (Pair) o;

        if (!a.equals(pair.a)) return false;
        return b.equals(pair.b);
    }

    @Override
    public String toString() {
        return "Pair(" + "a=" + a + ", b=" + b + ");"
    }
}
Simple class example: a Pair of A and B

class Pair {
    private A a;
    private B b;

    public Pair(A a, B b) {
        this.a = a;
        this.b = b;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || !(o instanceof Pair)) return false;
        Pair pair = (Pair) o;
        if (!a.equals(pair.a)) return false;
        return b.equals(pair.b);
    }

    @Override
    public String toString() {
        return "Pair(\"a=\" + a + ", \"b=\" + b + \\");
    }
}

IntelliJ can write most of this code for you automatically, then you tweak it.
What's special about java.lang.Object?

Every single class “extends” java.lang.Object
If you have a method that takes an Object, you can pass it anything

We can use this to write general-purpose reusable classes
Example: edu.rice.util.Log takes any Object, not just String
Internally, it just calls the .toString() method.

Coming soon: using Java “generics” instead of Object
Getting up to speed on Java

There are hundreds of Java books out there, fewer on Java 8
None of the Java 8 books seem to target beginners. We’ll do better.

There are web sites like StackOverflow
You’ll find good answers to specific questions
  • (e.g., “how do I work around the bug in Library X when it gives me some error?”).

You won’t find broad approaches discussed.
They won’t teach you computer science.
Resources you’ll use all the time

Oracle Javadoc web site
https://docs.oracle.com/javase/8/docs/api/
Documentation for all of the official support classes that come along with Java.

Google Java style guide
http://google.github.io/styleguide/javaguide.html
Explains things like indentation, whitespace, variable names, commenting. When in doubt, follow the advice here.

• We’re going to use CheckStyle, which enforces this style!
When in doubt...

1) See if somebody else already asked your question

2) Follow-up to an existing question before starting your own

3) Many questions have answers on the Internet
   You’re welcome to make any Google query you like, but please keep all questions related to Comp215 on Piazza.

   Posting a question elsewhere is an Honor Code violation.

4) All the labbies, TAs, professors answer Piazza
   You’ll get answers much faster than sending us email.
   And if you email us, your answer may well appear on Piazza.
Before you go to sleep tonight

Fill out the web form so we know you’re here.
http://goo.gl/forms/ArYKWBc0zY
(also linked from comp215.blogs.rice.edu)
  • If you did this earlier this summer, you don’t have to do it again

We’ll hand out project 1 on Wednesday, due Sunday night.