Welcome, welcome!

Java (really, Java8)

Recursion and recursive data structures

Software engineering (performance, testing, extensibility)

Some fun projects to stretch your brain
Java8?

You might have had Java in high school...
But the new Java8 features really change what you can do.
We’ll do things the “new” way whenever possible.
Getting started with Comp215

You need to fill out our online form
goo.gl/forms/84b4Gc5We6

But don’t panic, everything is online (that URL, slides, etc.)
comp215.blogs.rice.edu

Generally speaking
The blog (public web site) will have the course schedule, slides, and syllabus
Piazza (private for the class only) will have labs, assignments and discussion
Course syllabus

These slides are just an overview. The full details are online.

http://comp215.blogs.rice.edu/course-syllabus/
Grades

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

*No extra credit. But sometimes “cool points.”*

**Weekly labs: 20%**
Each of these will take about an hour and is either checked-off or not.

**Midterm exams: 30%**
Taken during class sessions, 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.
Grade disputes

Grades are posted on Owlspace
Log in and see your grades any time you’d like

Grade disputes must be submitted in writing, on paper
Include all the necessary information (code, etc.) to evaluate the dispute
Disputes must be submitted at most one week after grades are posted
Speak to Dan or Zoran offline before you submit if you want a gut-check
Weekly schedule

Lecture: MWF (Brockman 101, 9:00-9:50am)
We start on time. Lecture notes will be posted online.

Weekly labs
T: 4-5pm (MechLab 251), W: 3-4pm (MechLab 251), W: 7-8pm (Brockman 101)
    You *must* attend at least one lab in person.
    Any labbie can check you off, even if you don’t finish in the lab, but only in the same week.

Assignments: generally published Monday in class, due Sunday 11:59pm

Office hours: see online (Dan’s hours: W/F 10-11 after class)
Late work policy

Assignments roll out every week. Late work is not accepted.
0-24 hours late: 30% penalty
24-48 hours late: 60% penalty
>48 hours late: no credit given
If you have a personal issue or schedule conflict, let us know in advance.

In the front-half of the semester, the assignments build on one another
After the due date, we'll provide you with (bug free!) reference code
Our advice: submit what you've got at the deadline, then move on
Submitting things

You’ll be using the Subversion version control system.
You’ll “check out” our sample code from the Rice Subversion server.
You’ll “commit” your work back to the Rice server.

IntelliJ will offer to put new files under version control for you. Say yes.
Files are color-coded. If you have a .java file that’s red, then it’s not being saved.
Don’t submit the “.class” files that the Java compiler generates.

Commit your work all the time, as you go.
Assume your computer might explode in a ball of fire at any time.
Version control systems are the most important bacon saving device available.
Honor code issues

Formal written Honor Code policy is in the course syllabus.

Your work must be your own.
You may not copy code from another student in class.

We use sophisticated plagiarism-detection software.
If you submit somebody else’s work, we will detect it.
Getting your computer ready

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

You’ll need to install several things on your computer
IntelliJ IDEA 14 (the free community edition does everything you need)
https://www.jetbrains.com/idea/download/

Java8 SE 8u51 (or later) JDK (not just the JRE)
IntelliJ project settings

Many things you need to do to get things working for Java8.

Project 1 walks you through the process. We'll get Piazza set up ASAP and you'll find the assignment PDF there starting on Wednesday.
Java8 in a nutshell

We’ll eventually get into lots of cool Java & IntelliJ features
Lambdas and streams
Inheritance and subtyping
Generics
Unit tests (including “mocks”), coverage tests, performance tests
And maybe a bit of multithreading and concurrency

So why not some other programming language?
Java vs. Python (static vs. dynamic)

Java forces you to declare all of your types in advance. Python doesn't.
Java will catch mistakes at compile time. Python won’t fail until runtime.
But Java is way, way more verbose than Python.


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, functional, polymorphic, multi-threaded.
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 vs. C / C++ / Objective-C

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 are easier to get working, and when they fail, they fail more predictably.

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.
Hello, Java

In Java, every object is an instance of a class. What’s a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }

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

In Java, every object is an instance of a class. What's a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }
    public void scream() {
        System.out.println(string + "!");
    }
}
```

Member variable. Only visible “inside” the class.
In Java, every object is an instance of a class. What’s a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }
    public void scream() {
        System.out.println(string + "!");
    }
}
```

Constructor. Takes an argument (string), and initializes the member variables. this is a pointer to the newly allocated chunk of memory.
Hello, Java

In Java, every object is an instance of a class. What's a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }
    public void scream() {
        System.out.println(string + "!");
    }
}
```

Method. Anybody who has a pointer to a Hello object can call its public methods.
In Java, every object is an instance of a class. What’s a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }

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

Many, many system classes provided to help you do common things.
```
Hello, Java

In Java, every object is an instance of a class. What's a class?

```java
public static class Hello {
    private String string;
    public Hello(String string) {
        this.string = string;
    }
    public void scream() {
        System.out.println(string + "!");
    }
}
```

Typical usage:

```java
Hello hello = new Hello("Rice");
hello.scream(); // prints "Rice!"
```
Classes & packages

Classes put code near the data it operates on.

Packages help you group together related classes.

package edu.rice.list;

class List { ... }

File name = class name. Directory = package name. (edu/rice/list/List.java)
Visibility / Scoping

public: visible to the whole world
private: only visible to callers within the class
protected: like private, but also visible to subclasses (more on this later)
““ = package scope: only visible to callers within the package

Why? Visibility mechanisms build a wall between interface and implementation
# Interface vs. implementation

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implementation 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Set”</td>
<td>Bit Vector</td>
</tr>
<tr>
<td>“Queue”</td>
<td>Doubly Linked List</td>
</tr>
<tr>
<td>“Map”</td>
<td>Hash Table</td>
</tr>
</tbody>
</table>
# Interface vs. Implementation

<table>
<thead>
<tr>
<th>Interface</th>
<th>Implementation 1</th>
<th>Implementation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Set”</td>
<td>Bit Vector</td>
<td>Balanced Tree</td>
</tr>
<tr>
<td>“Queue”</td>
<td>Doubly Linked List</td>
<td>Balanced Tree</td>
</tr>
<tr>
<td>“Map”</td>
<td>Hash Table</td>
<td>Balanced Tree</td>
</tr>
</tbody>
</table>
Java interfaces

Interfaces go into .java files, look similar to Java classes

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

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

Warning: this is different from java.util.Set. More on that Wednesday.
Interfaces go into .java files, look similar to Java classes

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

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

*Warning: this is different from java.util.Set. More on that Wednesday.*

Interfaces have method “signatures” (argument, return types) but not “bodies”. (Yet.)
Interfaces go into .java files, look similar to Java classes

```java
interface StringSet {
    StringSet union(StringSet other);
    StringSet intersection(StringSet other);
    int cardinality();
}

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

Classes declare that they implement interfaces.

Warning: this is different from java.util.Set. More on that Wednesday.
Java interfaces

Interfaces go into .java files, look similar to Java classes:

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

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

**Old-school usage:**

```java
StringSet set = new StringTree();
set = set.add("Hello");
set = set.add("Rice");
```

**Equivalent new-school “pipeline” usage:**

```java
StringSet set = new StringTree()
    .add("Hello")
    .add("Rice");
```

**Warning:** this is different from java.util.Set. More on that Wednesday.
Java class inheritance

We’ll get very fancy with this later. For now, keep it simple.

Classes can *inherit code* from one another.

class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}
Java class inheritance

We'll get very fancy with this later. For now, keep it simple.

Classes can inherit code from one another.

```java
class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}
```

Simple so far

A a = new A("Arrgghhh");
a.say("Hello"); // prints: Hello
a.scream("Hello"); // prints: Hello, Arrgghhh
Java class inheritance

We'll get very fancy with this later. For now, keep it simple.

Classes can *inherit code* from one another.

```java
class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#");
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
}
```
Java class inheritance

We’ll get very fancy with this later. For now, keep it simple.

Classes can inherit code from one another.

class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#");
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
}

Supercall constructor
Java class inheritance

We’ll get very fancy with this later. For now, keep it simple.

Classes can *inherit code* from one another.

class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#"));
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
}

*Overridden method*  
*Compiler directive (optional)*
Java class inheritance

We’ll get very fancy with this later. For now, keep it simple.

Classes can *inherit code* from one another.

class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#"));
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
}

*Calls to B.scream() go here*
Java class inheritance

We’ll get very fancy with this later. For now, keep it simple. Classes can *inherit code* from one another.

class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#"));
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
    public void scream(String string) {
        say(string);
        System.out.println("Umm, " + string);
    }
}

These go to the “instance” type: B.say()
B “inherits” some methods from A and “overrides” others

```java
class A {
    private String epithet;
    public A(String epithet) {
        this.epithet = epithet;
    }
    public void say(String string) {
        System.out.println(string);
    }
    public void scream(String string) {
        say(string);
        say(epithet);
    }
}

class B extends A {
    public B() {
        super("&^!(@*#");
    }
    @Override
    public void say(String string) {
        System.out.println("Umm, " + string);
    }
}

B b = new B();
b.say("Hello"); // prints: Umm, Hello
b.scream("Hello"); // prints: Umm, Hello / Umm, &^!(@*#")
Zen Coding Rule #1: Don’t Repeat Yourself

Class inheritance lets you reuse code across classes.
.... Avoid the risk that you’re repeating a bug over and over again.

Less contrived example: java.lang.Object (every class extends Object)

    boolean equals(Object o) { ... }
    String toString() { ... }
    int hashCode() { ... }

Instead of StringSet, IntSet, FloatSet, ... just Set ... of Objects (for now)

    Set set = new Set().add(“Hello”).add(“Rice”);

Set implementation takes Objects, you can pass in anything.
Rule: if class B extends A and a method expects an A, you can pass a B.
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }
}
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
       Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }
}
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }
}

if it’s null, it’s not equal
Overriding methods on Object

Example: Overiding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }
}

if it’s not a Foo, it’s not equal
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }

    "type casting"
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }

    compare the internal strings
Overriding methods on Object

Example: Overriding equals()

class Foo {
    private String a, b;

    public boolean equals(Object o) {
        if (o == null || !(o instanceof Foo)) return false;
        Foo other = (Foo) o;
        return this.a.equals(other.a) && this.b.equals(other.b);
    }
}

IntelliJ can “generate” this sort of code for you with a nice wizard
Class vs. interface inheritance?

You can only extend one class, but you can implement many interfaces. But class inheritance includes code and data. Java8 also lets you put (limited) code in an interface. (More on that later.) There are other variants, like abstract classes, which we won’t ever use.

If you had Java in high school, you probably learned many things that you shouldn’t do in Comp215.

The guiding zen rule: don’t repeat yourself.
One last Java feature: static methods

Static methods can be called anywhere; no object instance. No inheritance / no overriding. Really, they’re just functions, hanging out with a class but not part of it.

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

The `main()` static method is where your program actually starts running.

Static methods also show up as helpers / utility functions.
There are hundreds of Java books out there, fewer on Java8
None of the Java8 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.
Before you go to sleep tonight...

Fill out the web form, so we know you’re here.
goo.gl/forms/84b4Gc5We6
(also linked at the course homepage: comp215blogs.rice.edu)

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