Lightning Round

instanceof

Used to test whether an object belongs to a particular type.

```
object instanceof TypeName
```

- evaluates to `true` if `object` is a `TypeName`, or if it is a subclass of `TypeName`
- evaluates to `false`, otherwise.

Doesn't depend on the static type of the variable, *only on the actual type of the object itself*

Trying out instanceof

```
Vector<Integer> v = new Vector<>();
ArrayList<Integer> al = new ArrayList<>();
List<Integer> ll = new LinkedList<>();
```

instanceof	v	al	ll
List	✅	✅	✅
ArrayList	❌	✅	❌
LinkedList	❌	❌	✅
Vector	✅	❌	❌

How to use instanceof

Can be good to "reclaim" some functionality through casting.

```
public interface Shape {
    public void draw();
    public void getArea();
}

public class Circle implements Shape {
    // just implements draw and getArea
}

public class Triangle implements Shape {
    // implements draw and getArea and...
    public boolean isEquilateral() {...}
}
```

How to use instanceof

Can be good to "reclaim" some functionality through casting.

```
List<Shape> l = new ArrayList<>();
l.add(new Triangle(3, 3, 3));
l.add(new Circle(4));
l.add(new Circle(5));

for (Shape s : l) {
    if (s.isEquilateral()) {
        System.out.println("Found one!");
    }
}
```

How to use instanceof

```
List<Shape> l = new ArrayList<>();
l.add(new Triangle(3, 3, 3));
l.add(new Circle(4));
l.add(new Circle(5));

for (Shape s : l) {
    if (s instanceof Triangle && ((Triangle) s).isEquilateral()) {
        System.out.println("Found one!");
    }
}
```

Safe! And functional.

Could also be useful for finding particular `Ship` objects, like `EmptySea`

hint hint

JavaDocs

What are JavaDocs?

Documentation generated by source code comments.

Useful for quickly writing thorough documentation for your projects, since you should be writing comments anyway!

- Can be written for classes, methods, and fields.
- Always placed immediately above the feature it documents
- Automatically generated using the `javadoc` tool.

How to Write JavaDocs

- Documentation is surrounded by `/**` and `*/`
- Each documentation comment typically contains:
  - Introductory text, the **first sentence of which is a summary statement**
  - A series of tags (prefaced by `@` characters)
- Within a documentation comment, you can use typical HTML tags (`em`, `code`, `img`, `ul`, etc.)

Tag	Description
@param <explanation>	One tag for each parameter of a method
@return <explanation>	The (explained) return value of a method
@throws <excepType> <explanation>	One tag for each exception the method throws, including explanations for what would cause it.
@deprecated	A warning to not use this method!
@author	Your name!
@see <other>	Reference another class, method, or field.

Example:

```
/**
 * If a part of this ship occupies this coordinate, and if the ship hasn't been
 * sunk, mark the part of the ship at that coordinate as "hit".
 *
 * @param row    the row of the shot
 * @param column the column of the shot
 * @return {@literal true} if this ship hasn't been sunk and a part of this ship
 *         occupies the given <code>row</code> and <code>column</code> and
 *         {@literal false} otherwise.
 */
public boolean shootAt(int row, int column) {
        ...
}
```

Generating JavaDocs

The full command:

```
javadoc [options] sourceFile1|packageName1|@fileList1|
                  sourceFile2|packageName2|@fileList2|...
```

There are also other options, like `-link`, `-d`, `-classpath`, `-sourcepath`, `-author`, etc...

Eclipse lets you manually configure many of these (I'm sure IntelliJ does the same, too.)

CIT 591 In Review

Things to Improve for Next Time

- Faster grading returns
  - better autograders on my end
  - better sense of the bottlenecks for each assignments
- Fewer assignment/slide typos
  - These slides were all new--thanks for helping me find the bugs
  - The assignments will continue to be made clearer and without typos
- Order of material? Overall set of concepts to learn?

Graphics in Java

java.awt

A Package for painting graphics and images.

Especially important: `java.awt.Graphics`, JavaDocs here

`java.awt` also has:

- Rectangles
- Polygons
- Points
- Fonts
- Shapes

javax.swing

A set of portable components that let you build windows and interactive elements on that work on nearly all platforms.

- Menus
- Layouts
- Lists
- Actions
- Buttons

Java GUIs in General

- GUIs are sets of components arranged in containers
  - Containers are components, and can be put in other containers
- Containers use layout managers to arrange elements inside of them
- Buttons, Lists, Menu Items, Checkboxes, and the like are available through `swing`
- General sketching and shape drawing goes inside of a **JPanel**, and is done using methods from `awt`

A Very Minimal GUI Program

```
public class SimpleSketch extends JPanel {
        private static final long serialVersionUID = 71485045288350360003L;

        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                var center = new Point(getWidth() / 2, getHeight() / 2);
                var radius = Math.min(getWidth() / 2, getHeight() / 2) - 5;
                var diameter = radius * 2;
                g.setColor(Color.WHITE);
                g.fillOval(center.x - radius, center.y - radius, diameter, diameter);
        }
        public static void main(String[] args) {
                SwingUtilities.invokeLater(() -> {
                        var panel = new SimpleSketch();
                        panel.setBackground(Color.GREEN.darker());
                        var frame = new JFrame("A simple graphics program");
                        frame.setSize(400, 300);
                        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                        frame.getContentPane().add(panel, BorderLayout.CENTER);
                        frame.setVisible(true);
                });
        }
}
```

Packages in Java

What Are Packages?

Sets of related classes, e.g. `java.lang`, `java.util`, `java.awt`, ...

The package a class belongs to is specified at the top of the class's file.

```
package pckg.name.with.periods
```

Importing Packages

`import java.util.Scanner` imports the `Scanner` class from the `java.util` package.

`import java.util.*` imports all classes inside the `java.util` package.

You don't need to import classes in the same package.

Importing Packages

If a package isn't imported, then its classes won't be visisble.

If the name of the class' package doesn't match the filepath that it's placed in, most IDEs will complain.

Naming Packages

I am sorry in advance for what you are about to see.

Naming Packages

Packages should have unique names.

Web domains have unique names.

"Let's name packages like websites, I guess. But we should write them in reverse."

Naming Packages

My email address is unique: `sharry@seas.upenn.edu`

My root package should be called, then:

```
edu.upenn.seas.sharry
```

I could then write other sub-packages:

```
edu.upenn.seas.sharry.hw1
edu.upenn.seas.sharry.hw2
edu.upenn.seas.sharry.calculus
```

Naming Packages

Finding a package based on its name:

`edu.upenn.seas.sharry.calculus` has its classes found in...

```
baseDirectory/edu/upenn/seas/sharry/calculus/
```

Regular Expressions

Regular Expressions At A High Level

Special text strings that match patterns in text.

They are technically limited in terms of what they can find, but they are very powerful for finding certain patterns:

- Good for matching phone numbers, email addresses, names, etc.
- Not good for matching Strings where there are more "a"s than "e"s.

Regular Expressions: A Brief Primer

- The empty regexp ϵ matches the empty string.
- Most characters as regexp match themselves.
  - *a* would match "a"
- If R_1 is a regexp and R_2 is a regexp, then $R_1 R_2$ is a regexp that matches a concatenation of whatever R_1 might match and then whatever R_2 might match.
- If R_1 is a regexp and R_2 is a regexp, then $R_1 | R_2$ is a regexp that matches whatever R_1 might match **or** whatever R_2 might match.
- If R_1 is a regexp, then R_1* matches 0 or more repetitions of R_1

Regular Expressions: A Brief Primer

Other shorthands:

- [] denotes a class of characters
  - [A-Z] is all letters A through Z, e.g.
- R_1+ matches 1 or more repetitions of R_1.
- . matches any single character.

Regular Expressions: A Brief Primer

An email address:

```
\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b
```

- \b ➡️ the beginning or end of a word
- [A-Z0-9._%+-]+
  - [A-Z0-9._%+-] ➡️ any character A-Z, 0-9, ., _, %, +, or -
  - + ➡️ one or more of a regexp from the previous group
- {2,} ➡️ two or more of the regexp from the previous group
- `\.` ➡️ a literal period character.

Regular Expressions in Java

Regexp themselves are implemented in the Pattern class.

The "search results" are implemented in the Matcher class.

Linked Lists

Here are some resources:

Node Data Structure

Linked List Implementations

Why Eclipse?

- It's free
- It's open source
- It's pretty ubiquitous
- It's extensible
- It doesn't require much of a license
- It's not too complex to learn

Why not IntelliJ?

In truth, when I do the class again, I'll just have the students use IntelliJ. That's what I use, anyway.

CIT 594 Preview