Recitation 2

Main Method, API & Packages, Java Basics

Demo 1: Making an application

Create a new eclipse project
- Eclipse: File -> New -> Java Project
- File -> New -> Class
- Check the main method stub. Hit "Finish"
- Write inside main method stub:
  - `System.out.println("Hello World");`
- Hit the green play button

Main method

When you run your application, it starts by calling method main:
```
public static void main(String[] args) { … }
```

Accepts one parameter of type
`String[]` (array of Strings)

Demo 2: Using program arguments

Now let's change the program to print out a user supplied argument!

Demo 2: Inputting program arguments

Now we'll tell Eclipse what arguments to use
- Run -> Run Configurations
- Click "Arguments" tab
- Enter arguments, and hit "Apply"

Exercise 1: Using program arguments

Write a program that prints whether a point is inside a circle. The program should receive 5 arguments in this order:
1. x coordinate of the point
2. y coordinate of the point
3. x coordinate of the circle's origin
4. y coordinate of the circle's origin
5. radius of the circle

Hints:
- Java arrays are 0-indexed
- `Double.parseDouble(str)` returns str as a double
- `Math.sqrt(d)` returns the square root of d
Java API & Packages

Java API (Application Programming Interface)

- Java provides a number of useful classes and interfaces
- The Java API documents how to use these classes. Each API page contains:
  - class/interface hierarchy
  - overview
  - methods
  - fields
  - constructors
- http://docs.oracle.com/javase/7/docs/api/index.html
- Also available on course website. Click the "Links" tab

Demo 3: How to use Java API

- Let’s make a program that takes a user supplied time (String) in the form of hours:minutes and prints out the hours and then the minutes.
- What class can help you with this?
  - Google search “Java 7 API <name of class>”
  - Click the docs.oracle.com link
  - Look for methods related to your task

Where did class String come from?

- Package java.lang
- Package: group of related classes
  - Can contain sub packages
- Why?
  - organization
  - namespace
  - encapsulation

Demo 4: java.lang is special

What happens when we try to use a class from a package other than java.lang?

- Make a method whose body is:
  - JFrame frame = new JFrame();
- Hover over the error and have Eclipse import the class
- Scroll to the top and see what the import statement looks like

Importing from other packages

- import javax.swing.JFrame;
- imports class JFrame from package javax.swing
- import javax.swing.*;
- imports every class and interface from package javax.swing
Exercise 2: Random numbers

- Write a function that accepts two parameters of type double, and prints out a random double between those two numbers
- Hints:
  - You will need to import a class from the Java API
  - Use your intuition about what class to use, and search Google for it
  - Look over the class’s methods to find one that can help you

Custom packages

- Except for the default package, file structure matches package structure
- Hard drive
  - Eclipse
    - Package Explorer
    - Eclipse Package Explorer
    - Other packages

Custom packages (continued)

- Importing works the same as the Java API
- Except for the default package, classes must declare their package above the class header

Java Basics

Primitive types vs classes

- Variable declarations:
  - int i = 5;
  - Animal a = new Animal("Bob");
- Animal and int are both variable types, but Animal is a class and int is a primitive type
Variables with a primitive type contain their value directly:

```java
int i1 = 5;
int i2 = 5;
```

So `i1 == i2` translates to `5 == 5`

Variables with a class type contain a reference to an object...

```java
Animal bob1 = new Animal("Bob");
Animal bob2 = new Animal("Bob");
Animal anotherReferenceToBob1 = bob1;
```

So `bob1 == bob2` translates to `Animal@0x36 == Animal@0x84`

While `bob1 == anotherReferenceToBob1` translates to `Animal@0x36 == Animal@0x36`

Default values

- What value does a field contain when it is declared but not instantiated?
  - `Animal a;` //null
  - `Object ob;` //null
  - `int i;` //0
  - `boolean b;` //false
  - `char c;` //\'\0\' (null byte)
  - `double d;`

Character class contains useful methods

- Examples of useful `Character` methods:
  - `Character.isDigit(c)`
  - `Character.isLetter(c)`
  - `Character.isWhitespace(c)`
  - `Character.isLowerCase(c)`
  - `Character.toLowerCase(c)`
  - see Java API for more!

These methods are `static` and are applied to `char c`

Demo 6: chars

- Notice the characters beginning with a `.\`. These are called escaped characters and have a special meaning
  - Examples: `\n` `\t` `\'` `\"` `\'`
  - Google search "java tutorial escaped characters" to see all the escaped characters
- Character int values for letters and numbers are sequential
- `chars` can be compared by their int value.
Strings: Special objects

- Strings are objects
- However:
  - They can be created with literals
  - They are immutable (unchangeable)

String literals

- String instantiation:
  - Constructor: `String s = new String("dog");`
  - Literal: `String s2 = "dog";`
  - Roughly equivalent, but literal is preferred

Strings are immutable

Once a String is created, it cannot be changed
- Methods such as `toLowerCase` and `substring` return new Strings, leaving the original one untouched
- In order to "modify" Strings, you instead construct a new String and then reassign it to the original variable:
  - `String name = "Gries";
  - name = name + ", ";
  - name = name + "David";

Strings are immutable

What happens when you execute this?
- `String name = "Gries";
- name = name + ", ";
- name = name + "David";

Strings are immutable

What happens when you execute this?
- `String name = "Gries";
- name = name + ", ";
- name = name + "David";
String concatenation

Operator + operator is called concatenation, or concatenation
- If one operand is a String and the other isn’t, the other is converted to a String
- Important case: Use “” + exp to convert exp to a String.
- Evaluates left to right. Common mistake:
  - System.out.println("sum: " + 5 + 6);
    - Prints "sum: 56"
  - System.out.println("sum: " + (5 + 6));
    - Prints "sum: 11"

Other String info

- Always use equals to compare Strings:
  - str1.equals(str2)
- Very useful methods:
  - length, substring (overloaded), indexOf, charAt
- Useful methods:
  - lastIndexOf, split, trim, contains, compareTo, startsWith, endsWith
- Look these up yourself in the Java API!

Key takeaways

1. The Java API is your best friend. Google search is a good way to find documentation on classes and methods:
   a. Other way to get to Java API: Course webpage, click “Link” in navigation bar, and click the Java API link.
2. Variables with a primitive type contain primitive values, those with a class type contain names (pointers to) objects, like String@45afbc.
3. Strings are immutable objects