**MODULE 1**

**Format**

- Fixes format of your code CTRL/Cmd + SHIFT + F

‐ Selects all code in Java file and fixes indentation CTRL/Cmd + A, CTRL/Cmd + I

**Getting Input**

• First, import the Scanner class: import java.util.Scanner;

• Create a scanner and assign it to a variable:

Scanner **scan** = new Scanner(System.in); // scan is a variable, can name it differently

‐ new Scanner(...) tells Java to make a new one (object) ‐ System.in tells Java that the scanner is to take input from the keyboard

• To read in the next int: int myNumber = scan.**nextInt();**

• To read in the next String: String myString = scan.**next();**

• To read in the entire next line as a String: String myLine = scan.nextLine();

**scan.close();** at the end

**Java Comments**

* Type the following /\* and then hit Enter, it will add a block comment
* Type the following right above a variable, method, or class name /\*\* and then hit Enter

**New Project**

Make sure public static void main(String[ ] args) IS checked

Make sure Inherited abstract methods IS NOT checked

Anything concatenated with a string is converted into a string

No string multiplication

Math.pow(x,4) //**power operation BECOME FLOATS**?

System.out.println("x pow 4: " + Math.pow(x, 4));

String string1 = “nn”;

String string2 = string1.toUpperCase();

String str1 = Integer.toString(int); //cast an int to a string

**String str = String.valueOf(num);** //another way to cast an int to a string DO NOT REMEMBER YET

String str2 = Double.toString(1.1); // cast a double to a string

int int1 = Integer.parseInt(“1”);// cast a string to an int

double db1 = Double.parseDouble(“1.1”); // cast a string to a double

**Getting a specific char in a String ,** use “charAt(int)”

String str = "cit";

char firstChar = str.charAt(0); //firstChar is ‘c’

myStr.**charAt**(u); refers to the index of the char in a string

**To convert a String to an array of chars,** use the ”toCharArray()”

String myString = “hello”; //arrayOfChars contains ‘h’, ‘e’, ‘l’, ‘l’, ‘o’

Char[] arrayOfChars = myString.toCharArray();

**To check whether a specified char value is a letter**, use “Character.isLetter(char)”

boolean isLetter = Character.isLetter('t'); //isLetter is true

**To check whether a specified char value is uppercase**, use “Character.isUpperCase(char)”

boolean isUpperCase = Character.isUpperCase('T'); //isUpperCase is true

**To check whether a specified char value is lowercase**, use “Character.isLowerCase(char)”

boolean isLowerCase = Character.isLowerCase('t'); //isLowerCase is true

**To convert a character to uppercase**, use “Character.toUpperCase(char)”

char myChar = Character.toUpperCase('t'); //myChar is ‘T’

**To convert a character to lowercase**, use “Character.toLowerCase(char)”

char myChar = Character.toLowerCase('T'); //myChar is ‘t’

**To compare characters** like you compare numbers using==, < , and > operators

char myChar1 = ‘s’; char myChar2 = ‘t’; boolean compared = myChar1 < myChar2; //compared is true

**A class includes**:

‐ Fields (instance variables) that hold the data for each object

‐ **Constructors that describe how to create a new object instance of the class**

‐ Methods that describe the actions the object can perform

public class ClassName {

// The fields (instance variables) of the object

String name; //declaration to store a String in the object t, defaults to null

double health; //declaration to store a double in the object

int age = 0; //declaration to store an int in the object, initially set to 0

// **The constructor for creating the object**

public ClassName(parameters) {

//code using parameters to set up **initial state of object**

}

**// A method for communicating with the object**

String getName(parameters) {

//returns value of “name” instance variable

//”this” refers to this instance of the class (ClassName)

return **this**.name;

}

}

public void withdraw(double amount) **throws Exception** {

if (amount > this.balance) {

throw new Exception("Amount is greater than available balance.");

}

**this**.balance -= amount;

}

public String getCustomerInfo() {

return **this**.customer.getName() + " from " + this.customer.getAddress();

}

try {

checkingAccount.withdraw(amount); //withdraw from checking

} catch (Exception e) {

//e.printStackTrace();

//prints custom error message from withdraw method

System.out.println(**e.getMessage**());

}

QUIZ 1

Question 8

If you want to define a char first, and then a String, what’s the correct type of quotation marks (and order) to use?



First “”, then ””



First ‘’, then “”



First ‘’, then ‘’



First “”, then ‘’

MODULE 2

**Testing for Equality in Java**

• Use == to compare primitives; boolean e = (2 == 3); //e will be set to true if 2 is equal to 3

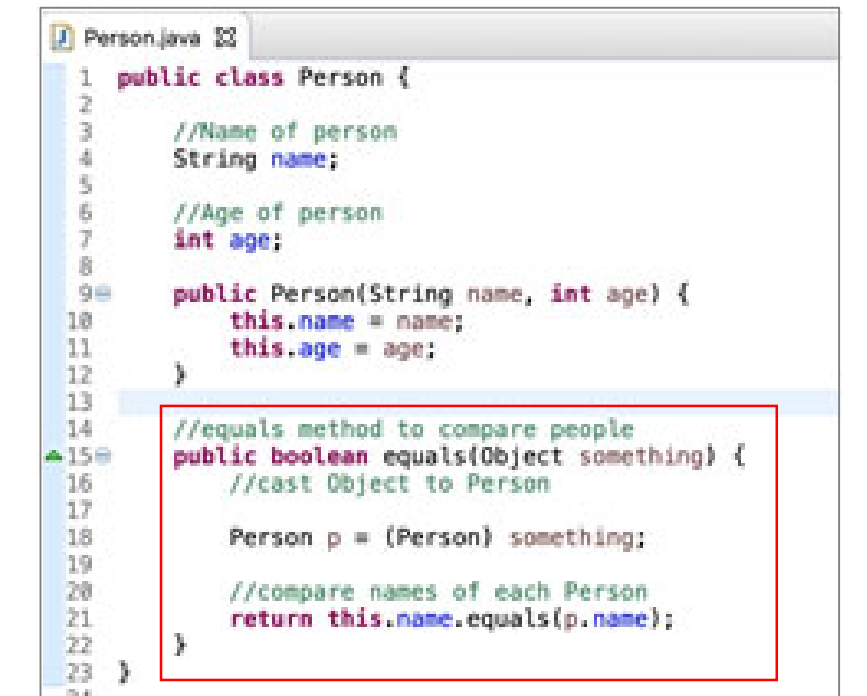
• Use the method x.equals(y) to compare Objects ;//e is set to true if “thisString” is equal to “thatString” boolean e = “thisString”.equals(“thatString”);

boolean e = “thisString”.equals(some**Unknown**String);

The JUnit method assertEquals(expected, actual) uses == to compare primitives and equalsto compare Objects

To define equalsfor your own objects, you’ll have to define exactly this method in your class: public boolean equals(Object obj) { ... }

‐ The argument must be of type Object, which isn’t what you want, so you must cast it to the correct type (e.g. Person)



**Terminology**

• A unit test tests the units (methods) in a single class

• A test case tests the response of a single unit (method) to a particular set of inputs

‐ You can (and should) have multiple test cases for a single unit test method

• An integration test is a test of how well classes and methods work together

‐ Integration testing (testing that it all works together) is not well supported by Junit and we won’t cover this

**Array**

You can't store different types in a single array

An array itself, is an Object

**Creating Arrays – 1st Step**

int[] myArrayOfInts;

String[] myArrayofStrings;

Customer[] myArrayOfCustomers;

**Creating Arrays – 2nd Step**

String[] names = new String[10]; //declare and create instance of array of 10 Strings

OR

int[] temps; //declare array

temps = new int[99]; //create instance of array with 99 slots

**Creating Arrays – 3rd Step**

Create an array of 3 doubles and sets the values

double[] myDoubleArray = new double[3];

myDoubleArray[0] = 5.0; //sets 1st value to 5.0

myDoubleArray[1] = 4.1; //sets 2nd value to 4.1

myDoubleArray[2] = 3.9; //sets 3rd value to 3.9

OR

String[] languages = {"Java", "C", "C++”}; //**one line** only otherwise illegal

Customer[] customers = {new Customer(“Brandon”), new Customer(“Betsy”)}; //one line

length is an instance variable, not a method (array.length in for loop)

On the other hand, Strings have a length() method

**Arrays cannot be easily resized**: You’d have to create a new array, copy everything from the old array, and add the new elements to the new array

**Enhanced for Loop**: Here we iterate over an array of Strings, where n represents each item in the array

for (String n : names) {

System.out.println(“Name: “ + n); }

// This simple structure allows you to visit each element of an array without explicitly expressing how to go from element to element using an index

**Default Value**

0 for non floating‐point types (e.g. int, byte, short, long) 0.0 for floating‐point types (e.g. double, float) **false** for booleans

the default value of an object variable variable is **null** • null is a legal value, but there isn’t much you can do with it ‐ It’s an error to refer to its fields, because it has none ‐ It’s an error to send a message to it, because it has no methods ‐ null is basically a pointer that doesn’t point to anything ‐ It’s very similar to Python’s None

**Copying Arrays : CANNOT use assignment**

• Array assignment (=) does not copy array values ‐ This is equivalent to the concept of assignment by reference in Python

• Given an array, we can not copy it directly

//Define array a int[] a = {1, 8, 3};

//Create an array b of same size as a **int[] b = new int[a.length];**

//Set b = a

//This does NOT copy elements of a to b //**It only makes b refer to same array object [1, 8, 3] b = a**;

• How do we know? Use == to compare object references System.out.println(a == b); //true

**Copying Arrays – Copy Elements**

• You can, however, create a new array and copy the elements directly

//Define array a

int[] a = {1, 8, 3};

//Create an array b of same size as a

int[] b = new int[a.length];

//**Copy elements of a to b using a for loop**

for (int i = 0; i < b.length; i++) {

b[i] = a[i];

}

• Use == to compare the objects System.out.println(a == b); //false

• And use the Arrays.equals method to compare the actual array contents (values) System.out.println(Arrays.equals(a, b)); //true

**Copying Arrays ‐ Cloning**

• You can also clone (create an exact copy of) an array using the clone method ‐ Many Java Objects support cloning

//Define array a

int[] a = {1, 8, 3};

//Copy elements of a to b

**int[] b = a.clone();**

• Use == to compare the objects System.out.println(a == b); //false

• Compare the actual array contents (values) Property of Penn Engineering | System.out.println(Arrays.equals(a, b)); //true

**Array of Arrays**

• Elements of an array can be arrays themselves

• The following creates an array of 3 arrays, each of which points to an array of 2 ints

int[][] table = new int[3][2]; //3 rows 2 columns

• **Then populates each slot in the array with an incremented count**

int count = 1;

for (int i = 0; i < table.length; i++) { //get the length of the rows (vertical)

for (int j = 0; j < table[i].length; j++) { //get the length of the columns (horizontal)

table[i][j] = count++; //set count value in each array slot, then increment

}

}

• This is like a “table” of 3 rows and 2 columns ‐ table.length is 3 ‐ table[0].length is 2

USING

import java.util.Arrays;

System.out.println(Arrays.deepToString(table));

//OUTPUT

//[[1, 2], [3, 4], [5, 6]]

This is the same as defining and directly populating a 2‐dimensional array like so

int[][] table2 = { {1, 2},

{3, 4},

{5, 6} };

• To compare 2 nested arrays, you can use the Arrays.**deep**Equals method to do a deep comparison System.out.println(Arrays.deepEquals(table, table2)); //true

**Non‐rectangular arrays**

Here we update the 3rd element in the array to point to an array of 20

ints table2[**2**] = new int[20]; //**[2] = row 2 compare int[][] compared below**

This is the same as initializing a 2‐dimensional array like so

int[][] table2 = {

{1, 2},

{3, 4},

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

};

**To access the value in any slot in a 2‐dimensional array**, specify the index number of the **row** inside brackets [], followed by the index number of the **column** inside brackets []

To print the value in row 0 and column 0, you’d use: System.out.println(table2[0][0]); //prints 1

To print the value in row 2 and column 2, you’d use: System.out.println(table2[2][2]); //prints 0

**An ArrayList**

• ArrayLists are not defined with a fixed number of slots – they have a variable length

• ArrayLists can only contain Objects and you can't store different types in a single ArrayList

• ArrayLists are part of Java’s Collections Framework

‐ Collections are defined in java.util

‐ To use ArrayLists specifically, you have to import java.util.ArrayList

‐ All Collections share similar methods (add, remove, size, etc.)

• Since you can’t store primitive values in an ArrayList, you have to use the wrapper classes associated with primitive types ‐ For int, use Integer; for double use Double, etc:

ArrayList <Data Type> myArrayList = new ArrayList<Data Type>();

ArrayList<Integer> numberList = new ArrayList<Integer>();

ArrayList<String> stringList = new ArrayList<String>();

ArrayList.size()(a method, not a variable) = = array.length (length is an instance variable, not a method)

ArrayList scores = new ArrayList();

scores.add(23); //adds element 23

scores.add(15); //adds element 15

scores.add(0); //adds element 0

System.out.println(scores.size()); //size of 3

ArrayLists have many attributes/methods : There is add, remove, size, get, etc.

Printing Objects

public class Customer {

//Name of Customer

String name;

public Customer(String name) {

this.name = name;

}

//toString method must return a String

public String toString() {

return **this.name**; //return name, to be printed by Java

}

}

Customer c = new Customer(“Brandon”);

System.out.println(“c = “ + c); //prints “c = Brandon”

**Static Variables**

**instance variables** == normal; can be different for every instance of a class, a true variable. Need an instance of the Class to access the instance variable.

**static variables** == constant, UPPERCASE, often refer to properties that are common to all instances of Class; no Need an Instance to access STATIC\_VARIABLE, use name of Class.STATIC\_VARIABLE; used for hard-coded value; static final;

**Static Methods**

No need to create an instance of a class to call a static method, use the class name to call the method Often times, Java “helper” methods are static

‐ Helper methods are utility methods that assist a program in doing some basic error checking or processing of a given input, for checking the validity of a number

**public static void main** is the first method Java looks for when running a program; it looks for a static main method in HelloWorld and runs it without creating an instance HelloWorld

- When there is no need for the method to belong to an instance of the object

- static variables can be used for constant Rule: Instance methods can access static variable

- another common usage is to use static variables **to share data across instances of an object**, e.g. to keep track of the object instances created

public class Car {

//list of all created cars

static ArrayList<Car> CAR\_LIST = new ArrayList<Car>();

public Car() {

Car.CAR\_LIST.add(this); //create car and add to list

}

public static void main(String[] args) {

Car car1 = new Car();

Car car2 = new Car();

for (Car car : Car.CAR\_LIST) {

System.out.println(car);

}

}

}//Don’t quite understand

Other Rules For Static

• A static method can access only static variables

‐ It cannot access instance variables

• A static method can call another static method

• An instance method can call a static method or access a static variable

• The keyword “this” does not make any sense inside a static method

**a signature is what distinguishes one function or method from another**

• In Java, two methods have to differ in their names or in the number or types or sequence of their parameters

• In Java, a method signature does not include the return type

**Polymorphism** many (poly) shapes (morph)

• In Java, polymorphism often refers to the fact that you can have multiple methods with the same name in the same class

Polymorphism is divided into two types:

‐ Overloading • Having two or more methods with the same names but different signatures

‐ Overriding • Replacing an inherited method with another having the same signature

So you can supply defaults for the parameters:

So you can supply additional information:

When you overload a method with another, very similar method, only one of them should do most of the work:

If your methods are legally overloaded, Java will figure out which one you want to use

constructor can call another constructor in the same class, but there are rules

‐ You call the other constructor with the keyword this

‐ The call must be the very first thing the constructor does

public class Point {

int x; int y; int sum; //creates a point at given x and y

public Point(int x, int y) { this.x = x; this.y = y; this.sum = x + y; } //creates a point at 0, 0 public Point() { this(0, 0); } }