Zachary D Clawson

CSC 116 – Test 2 Study Guide

Class variable – a method which belongs to the entire class
public static double circleRadius;
Instance variable – a method which belongs to an instance of the class
private double radius;
may belong to a particular circle

Lecture 7 and Lecture 8

Boolean logic – logic that evaluates to either true or false
Boolean operator - >, <, <=, >=, !=, …
Boolean variables – boolean x = true or false; OR we can do boolean x = 8>5; ...something like that
and operator – for boolean logic: &&
or operator – for boolean logic: ||
exclusive or operator – for boolean logic: ^
not operator – for boolean logic: !. EXAMPLE: !(5>2) is FALSE

Lecture 9

switch statement – breaks a variable down into cases of occurrences
- after each case use word "break;"
- "default" is used to describe cases that aren't listed

Lecture 10

Iteration – when a program needs us to do something over and over again
Loops – used to do repetitive iterations of some code until a condition runs out
- 3 types:
  - while
  - do-while
  - for

while Loops – we want to do something over and over again but we don't know how many times, if at all. EXAMPLE: while(x<10){x = x+1;}
do-while Loops – we want to do something over and over again, we don't know how many times, but we want to do it at least once. EXAMPLE: do{x=x+1;}while(x<5);
for Loops – we know exactly how many times we want to execute a code
EXAMPLE: for(int i = 0; i<5; i = i + 1){System.out.println(i);}
break (in loops) – can use in a while or do-while loop in order to break if we exceed a certain condition (use an if statement inside)
continue – stops the current iteration of a loop and restarts
Increment operator – want to add 1 to **i** in each iteration? Use i++ if you want to use the i first in something, and then add 1
EX: i=0; System.out.println(i++); prints 0 then adds 1 to I
Use ++i if you want to add 1 before i is used (would print out 1 in above example)
Decrement operator – i-- does the same thing as i++

Lecture 11

Class – an abstract representation of something

Zachary D Clawson

Object – a specific instance of a class
static – this word used with methods/variables tells Java that we want them associated with the class. If not used then we're telling it we want it associated with a particular instance of the class

creating a new object – write *Example ex = new Example();* to create a new object called ex in a class named Example. If we create an instance variable: *public double x;* we can assign *ex.x = 5;* a value to the instance variable of the object.

Attributes – instance variables referred to in the context of objects

what if we create two objects, but set the second equal to the first?

- the first object created: *Car car1 = new Car();*
- the second object is created: *Car car2 = car1;*
  - references *car1*
  - this actually just points to the same location in memory as *car1*, so if we change an attribute to *car2* it will change it for *car1* as well, and vice versa
  - this is called creating a second **reference** in memory to this object

non-static methods

- we can create non static methods to be used with objects
- just create a method without the word static and then use the instance variables
- then when referencing it, just do it as: *object.method();*

object creation

- just saying *Car myCar;* declares the object as **null**. We could also set it equal to **null**.
- when we set the object = *new Car();* the object is created **created when **new** is used
- common error is **null pointer exception** where object is just declared as **null** then used

Constructor – gives an object's instance variable and initial value; executed automatically when object is created
EXAMPLE: *public Car() {tankSize = 10.0;}*
We can overload constructors so we can assign values if we want. We could ADD to the above code: *public Car(double size) {tankSize = size;}*

- if we create a local variable in a constructor with the same name as an instance variable used, we use the keyword **this** to point to the instance variable
- ALWAYS DECLARE INSTANCE VARIABLES AS PRIVATE

Lecture 12

Setter – used to set a variable to a specified value
Getter – used to get the variables current value

- setters and getters and useful when instance variables are private so external classes can reference and use them

printing an object

- *System.out.println(myCar);* prints out something like *Car@junkcode*
- in order to do this, make a non-static method called *toString()* which returns a String with

information about the object

objects as parameters

- can use objects as parameters in methods
- then use setters/getters (or just reference the instance variables: *object.instanceVar = 5;*) to modify object

object equality

- equals operator == does not check for equality, checks to see if two objects reference the same space in memory
- use .equals() method to test whether two points are actually equal
- method will return boolean true or false
- first, method tests if Object o is an instance of the class

EXAMPLE

```
public boolean equals(Object o)
{
	if(o instanceof Point)
	{
		Point temp = (Object)o;
		if(temp.getX() == this.x && temp.getY() == this.y)
		{
			return true;
		}
	}
	return false;
}
```

In code, you'd type: *p1.equals(p2);*

Lecture 13

Package – collection of classes that all have something in common (such as they are all used for input or output). EXAMPLE: Math or String

importing packages

- *import java.util.Scanner;*
- *import java.io.*;*

try block – must put file writing / opening / declaration / printing in a try/catch block
FileWriter class – handles opening a file for writing (it will create one if it does not exist and overwrite if it does)

- *FileWriter fw = new FileWriter("example.txt");*

PrintWriter class – handles writing to the file, using it as a parameter

- *PrintWriter pw = new PrintWriter(fw);*

closing the file – if you open a file for writing (using FileWriter), you must close it

- *fw.close();*

errors that occur

- can occur when we open or close file

- so make two try statements, one for each opportunity
- to do this we must declare the file we are writing to as null before the first try block
- *FileWriter fw = null;*

PrintWriter pw = null;

Lecture 14

Scanner can process Strings and Files

- *Scanner scan = new Scanner(stringName);*
- *Scanner scan = new Scanner(fileName);*

Tokens – individuals words or numbers in a File/String

what if there is an error in our text file we're reading / processing?

- first use a scanner to read in the file (inside a try block)
- within that try block, start your while loop to execute while there is a line to be read
- then create a String which accepts the whole line
- assign the String with the Scanner class and read each token at a time

Lecture 15

Immutable – once its created, it cannot be changed

- Strings are these
- each time we change a String, java just creates a new one

Index – each character has a position in the string, known as an index

- first character is 0

Substring – String within another String

String methods

- format as *stringName.methodName();*
- *char c2 = stringName.charAt(2);*
- *int x = s.indexOf("er");*
  - returns 2 if *String s = "over there";*
  - prints -1 if substring does not occur in String at all
- *String up = s.toUpperCase();*
- *String dn = s.toLowerCase();*
- *String sub = s.substring(6);* → prints everything starting at index 6 and to the right
- *String sub = s.substring(2,6);* → prints everything starting at index 2 and ending at 5

format String method

- basically build a String which has text formatted the way you want, with a **%** sign to denote where data falls
- %d – integer
- %f – decimal / floating pt number
- %s – String
- %c – Single character
- EXAMPLE: *String formatString = "%s %d %s";*
  - formats a string so that it will have String, integer, String with a space between each

- then we would do something like:
  - *String output1 = String.format(formatString, "Joe", 16, "Male");*
- we can add a number between the % and the letter to specify how many spaces we want to allot for the word
- automatically right-aligned, but if you put a – before the number, you get left alignment
- can also specify number of decimal places. EXAMPLE of 2 decimal places for a number taking up 8 spaces: *%8.2f*