Name _____ Date: ___/___/____
Mr. Merlis – Java 2 Day of Class: __1__

REVIEW OF WHAT WE SHOULD ALREADY KNOW

THE JAVA PROGRAMMING LANGUAGE

OOP what?

Java is an object oriented programming language (OOP.) Wikipedia defines this to mean that a computer program may be seen as composed of a collection of individual units, or *objects*, that act on each other, as opposed to a traditional view in which a program may be seen as a collection of functions or procedures, or simply as a list of instructions to the computer.

How do we make Java programs?

By writing programs (the source code) using Java's built-in syntax and library files (found in the API.) To actually run the program, we must first compile it, which ensures the correct syntax usage. If you are missing a brace or spell a Java keyword incorrectly, the compiler will pick it up and tell you. This is called a **syntax error**.

If the compile is successful, the source code is translated into Java **byte code.** Byte code is the "machine language" for the **Java Virtual Machine** (JVM), which functions as though it is a computer. It **interprets** the byte code and runs your program. The process we should follow is *write* the source code*, compile* frequently*, execute* when testing and done.

So how can write these programs?

The "only" thing we have to do is write the source code. Since Java is object oriented, all source code must be contained within **classes** from which **objects** can be **instantiated**.

Classes

We write our code inside of classes. Classes store the information about an object in the form of its description and what it can do. In other words, these are adjectives and verbs. If we were designing a class called "Car", we might describe the car by saying its make, model, and year. The car may be able to drive forward, drive backwards, and fill its gas tank. The descriptors are called **instance variables** as they can be different for each *instantiation* of the Car class*.* The driving and filling of the gas tank are actions, and they are called **methods** in Java. We may also see the word "fields" used in place of instance variables, so it is said that classes are composed of fields and methods.
(We are leaving out class variables right now; more specific definitions will come.)

Types of Classes

There are two types of classes we will be working with: application programs, and object classes. Object classes store all the information for objects that can be made (fields and methods) while application programs use objects built from those classes to perform tasks (and literally drive the program.) For this reason, application programs can also be called **application program drivers**. Application programs must **always** have a main method.

Problems to Think About…

1. What would happen if you tried to compile a program that was missing a semi-colon at the end of a line of code?
2. What is it called when you make an object from a class?
3. Which type of class must always have a main method?
4. What two things are object classes composed of?

WRITING YOUR FIRST PROGRAM

- To start a class, you simply write:

public class *SampleClassPerson*
{ → Note that you will need a corresponding closing brace at the end of your program.

- Next, we list all of our instance variables.
  private String myName;
  private int myAge;
  private boolean likesCandy;

Note that ALL instance variables are private!
This means that they can ONLY be "seen" or accessed from within this class.
To create an **instance variable**, you write the word "private", then the variable's type, followed by the variable's name.

The variable's name is called an **identifier**, and this applies to any name you give a method, variable, or class.
All identifiers should be written in **titlecase*,** which means that the first letter of every word is capitalized **except** for the first. Note that identifiers should have NO SPACES. In other words, even if you want to call it several words, they should all be put together like you see above.
*Class names are an exception; the first letter of every word, including the first, should be capitalized.

CREATING VARIABLES

When you create a variable, you need to give it a **type** and a name. This is called **declaring** the variable. At some point you should **initialize** the variable, which means give it a starting value.

If the variable is an instance variable, initialize it in the constructor.
If the variable exists within a method, you should initialize it at the same time you declare it.

What's a *TYPE*?

A type specifies the kind of variable you want to use. There are three primitive types we will use (int, double, and boolean). Any other "type" of variable must be an instantiation of an object. Familiar types to us should include ArrayList and Random. Notice that when you make variables of those two types, you must not only declare them, but then call their constructors to set up the room in memory for these objects to work their magic.

*String IS an object, but is so commonly used that we don't have to call its constructor, Java does it automatically for us when we initialize a String object.

What's a *CONSTRUCTOR*?

A constructor is a method that has the same name as the class it is in.
Based upon our example above, this class would have a constructor that looks like this:

public *SampleClassPerson*(*list of parameters)* → header for the constructor [the first line before the "{"]

The purpose of the constructor is to initialize the private instance variables.

- And finally we write the methods for this class. Most of these will be public so that other classes (in particular, the application program) can use them. You may want to include "accessor methods" which simply return the values stored in the private instance variables.

---

5. What is it called when you give a variable a name and a type?

6. What mathematical sign do we use to initialize a variable?

7. What method in a class has the same name as that class?

8. Why are accessor methods necessary?

A SAMPLE OBJECT CLASS

```
/**
* This is called a Java Doc Comment.
* It is extremely useful because it will generate an API.
* An API (application programming interface) will display the headers of the methods of your class as well as
* any description that you supply.
* @author Mr. Merlis → That will put your name.
* @version 1/31/2006 → Put the date!
* There are other ways to display information, but at the least we should always include your name and the date.
*/

public class Person
{
	// these are the instance variables
	private String myName;
	private int myAge;
	private boolean likesCandy;


	//---THE METHODS ARE BELOW----

	// constructor that takes zero parameters
	public Person()
	{
		myName = “John Doe”;
		myAge = 0;
		likesCandy = false;
	}//====================================

	// constructor that takes the parameters
	// from the user of the program
	public Person(String name, int age, int lCandy)
	{
		myName = name;
		myAge = age;
		likesCandy = lCandy;
	}//====================================

	// returns the person’s name
	public String getName()
	{
		return myName;
	}//====================================

	//returns the person’s age
	public int getAge()
	{
		return myAge;
	}//====================================

	//returns true if the person likes candy
	public boolean likesCandy()
	{
		return likesCandy;
	}//====================================

	// returns the objects info as a String
	public String toString()
	{
		String nameAge;
		nameAge = (myName + “ is “ + myAge);

		if(likesCandy)
			nameAge += “ and does “;
		else // doesn’t like candy
			nameAge += “ and does not “;

		nameAge += “ candy.”;

		return nameAge;
	}//====================================

}// end of class
```

UNDERSTANDING OUR SAMPLE OBJECT CLASS

9. How many instance variables are there?

10. How many methods are there?

11. What are the accessor methods?

12. Why do we typically give instance variables the prefix "my" or "its"?

13. Could we have more than three accessor methods?

THE CONSTRUCTORS

There are two constructors. One takes zero parameters, the other takes three.
→ *Notice* the similarity between the code in both of them; in each case, they are initializing the instance variables!
→ *Notice* that the zero parameter constructor gives the instance variables default data; hence the **default constructor**.

THE METHODS

Our class has three accessor methods and one *toString* method.
→ *Notice* the return types of each method. (The word that follows public.)
→ *Understand* why a return type is necessary.

Just what, exactly, is a *toString* method?
It should always have this header: **public String toString()**

A toString method is necessary in order to print out specific information about an object. We must remember that Java is an OOP, such that every time we instantiate an object from a class, it is given a place in memory. By default, the computer doesn't really know what kind of object it is; all it knows is that it **is** an object and it **has** a place in memory. If you were to create a new person in an application program using the following line of code:

Person tomU = new Person("Tom Urbanski", 16, "yes")

and then write the following line of code:

System.out.println(tomU);

If we do **not** have a toString method, it is going to print something that looks like this: **Person@48FAC1**

The reason it does this is because whenever **System.out.println** takes an object as a parameter, it looks for a toString method. The class **Object**, itself, has one, which prints out the class name followed by the hash code (place in memory.)

→IMPORTANT←
All classes extend Object, even if you don't write it!
Therefore it will use the toString method in the Object class unless you overwrite it!

By putting a toString method in your object class, **System.out.println** will use it, printing the information about the object exactly the way you want it to appear. This makes it very convenient to print out all the objects stored in lists because we can use for loops and other methods to 'iterate' (loop) through them.

---

14. Why should we always include a default constructor?

15. What method should we **always** include in our object classes?

USING OBJECTS

The purpose of object classes is to create objects capable of performing specialized tasks for us in a program. This goes back to the definition of **OOP**, whereby rather than have a linear progression through one "app program", we simply use the app to drive everything forward, calling out to different, specialized classes where all of the "nitty gritty" occurs.

To use an object from a class we created, we must declare and instantiate it by using the following syntax:

ClassName nameOfThisObject = new **ClassName**(); // if the constructor takes parameters, include them!

Understanding the line above…
→ The left side of the equal sign is the *declaration* of the object, whereby you give it a name and a type.
→ The right side of the equal sign is the initialization, whereby you say *make me a new object of this className*.

Now that we have an object from this particular class (**className**), this object is capable of carrying out any method in its class definition. (That would be the class it came from.) If you want to use a method from this class, write the following:

nameOfThisObject.nameOfTheMethod(); // if this method takes parameters, include them!

Understanding the line above…
→ The left side of the "." is the executor; which is typically the object.
→ The right side of the "." is the name of the method.

MORE ABOUT PARAMETERS…

When **writing** a method, it must have a **header** and a **body**.

```
public int addThreeNum(int a, int b, int c) // the header
{ // ← that brace and below are the body
      return (a + b + c);
}//=====================================================
```

Remember
- In the header, the parameter must explicitly be given a **TYPE** and a **NAME.**
- If the return type of the method is NOT void, then you **must** return something.
- If you try to return something that is **not** the correct return type, you will get an error.

When **calling** a method, you do **NOT** need to write the type of each parameter.
Example: `executor.addThreeNum(14, 16, 15); // returns 45`

Notice: All we had to do was give parameters of the correct type.

USING NON-VOID METHODS

You will often use methods that return things, such as a boolean method (returns true or false) or some other type of method. More often than not, you should declare a variable that will hold the value returned by the method call.

Application

In the box above, we made a method that adds three numbers. For this method to be helpful, however, we should create a variable to hold the sum, which can later be used to make program decisions.

```
int sum = executor.addThreeNum(14, 16, 15);
```

When you are working with Random objects, it is especially important to place their values in variables.

---

16. What do you think happens if you try to call a method with parameters of the wrong type?

17. Why are there parentheses at the end here: Turtle sam = new Turtle();

A SAMPLE APPLICATION PROGRAM

```
import java.util.Random;      // the Random class
import java.util.ArrayList;   // the ArrayList class

/**
* This is a sample application program.  Make sure you understand every last part of it! :)
*
* @author Mr. Merlis → That will put your name.
* @version 1/31/2006 → Put the date!
*/

public class PersonApp
{
       public static void main(String[] args)
       {
              ArrayList group = new ArrayList();    // an ArrayList with the name 'group'        - 1
              Random rand = new Random();           // a Random object with the name 'rand'      - 2

              int numPeople = rand.nextInt(10) + 1; // a random number between 1 and 10          - 4
              boolean useDefaultCon = false;        // ALWAYS INIT. bool. var. @ time of dec.    - 5

              // variables below will randomly get values when using the 3 param. constructor   - 7
              int itsAge = 0;                                                                    - 8
              boolean likesC = false;                                                            - 9

              // loop below adds numPeople people to the ArrayList                              -11
              for(int w = 0; w < numPeople; w++)
              {
                     useDefaultCon = rand.nextBoolean();  // randomly returns true or false     - 14
                     if(useDefaultCon)                    // use the default constructor
                            group.add(new Person());
                     else
                     {
                            itsAge = rand.nextInt(81);    // random # between 0 and 80
                            likesC = rand.nextBoolean();  // true or false
                            group.add("Jesse Smith", itsAge, likesC);
                     }
              }// ends the for loop                                                            - 23

              // This loop prints out the people who like candy
              for(int j = 0; j < numPeople; j++)                                             - 26
              {
                     if( ((Person)group.get(j)).likesCandy() ) // when the person likes candy
                     {
                            System.out.println((Person)group.get(j));
                     } // ends the if
              }// ends the for                                                                 - 32

              //----------THE 7 LINES OF CODE ABOVE COULD BE WRITTEN WITHOUT BRACES!
/*----------------------
--            for(int j = 0; j < numPeople; j++)
--                   if( ((Person)group.get(j)).likesCandy() ) // when the person likes candy
--                          System.out.println((Person)group.get(j));
--
By default, if braces are not included in an IF statement or FOR loop (and in many other settings we will soon
learn) then the FIRST line of code after the closing parenthesis (for the IF or FOR) is considered part of that
control structure.  (It will look for the first semi-colon.)
*/
}
```

---

18. What is the value of *count* that is printed at the end?

```
int count = 0;
for(int w = 0; w < 4; w++)
      System.out.println("adding one to count");
      count++;
System.out.println("Count now equals: " + count);
```

19. Where above was casting necessary? Why is it needed?

UNDERSTANDING OUR SAMPLE APPLICATION PROGRAM

THE IMPORT STATEMENTS

Java provides us with a library of over 1,000 classes we can use in our programs. In order to use these already-made classes, we must import them. We will talk more about the API in class.

THE VARIABLES

Six variables belong to this main method, with two more used locally in the FOR loops.
Lines 1 and 2 create the ArrayList and a Random object.

Line 4 declares a variable that takes the value of the number of people to add to the list.
The boolean variable, useDefaultCon, comes into play later. Notice how it is INITIALIZED to **false**, when constructed.

The last two variables are preceded by a comment that explains their purpose. Notice that they are also given names that *make sense*! This makes the programs more readable.

THE FIRST FOR LOOP

This loop is a bit tricky; it will loop *numPeople* times, and add a new person to the list each time.
Notice, however, that it will randomly either add someone using the default constructor or the one that takes parameters.

Line 14 randomly gives the variable useDefaultCon a value of true or false.
→ If it is true, then line 15 is true, resulting in the execution of line 16.
→ If it is false, then lines 19-21 are executed, randomly coming up with an age and if the person likes candy.
Line 21 adds this person to the ArrayList.

THE SECOND FOR LOOP

This loop is "special" in that it only prints out those who like candy. Therefore, its first line is an IF statement to determine if this particular person likes candy. Let's take a closer look:

```
if( ((Person)group.get(j)).likesCandy() ) // when the person likes candy
{
                System.out.println((Person)group.get(j));
} // ends the if
```

This looks confusing!
Yes, it does! Let's understand what is going on.

An ArrayList stores OBJECTS. They can be of any type, in fact, an ArrayList can have completely different objects in each slot. Therefore, if we simply wrote: group.get(j).likesCandy(), we would get an error saying that no such method exists, because **group.get(j)** would return an object that think it is "just" an object. The **Object class**, as written by the creators of Java, does **not** have a *likesCandy* method.

To tell the computer that this is a **Person** object, you must **CAST** it by writing the name of the class in parentheses in front of the call that returns the object.

All those parentheses!

→ **((Person)group.get(j)).likesCandy()**

We need to cast the object to Person **before** it tries to execute the likesCandy() method. This explains the order of the parentheses; it comes down to order of operations.

THE PRINT STATEMENT - `System.out.println((Person)group.get(j));`

Since we have a toString method in the Person class, the System.out.println call will print the information for the person stored in the "j-th" slot of group. (Remember that group is the name of the ArrayList!)

IN SUMMATION

Computer Programming is all about problem solving. Once you have mastered this short review of what you should already know (or better learn quickly), you will have the tools to solve any problem presented.

The best way to solve a problem is by first thinking it through! Don't try to code it so much as think of the logic involved; say it out loud in simple English! Devise a plan, and then think in terms of using Java syntax to find the solution.

It is very important when you program to use correct style in your programs. Always indent new blocks of code, whether explicitly after a brace or as one line of an IF statement (when not using braces.) Give your identifiers logical names and do not take short-cuts. Be precise!

Use comments! BE SURE to include Java Doc comments as well. You needn't comment every line of code, but comment when you feel necessary and appropriate. It is good programming practice to comment before any block of code to explain what it does. If it is a particularly tricky section of code, it would also help to comment a little inside of the block at times you find critical to understanding the logic.

Use the API!

Understand your errors! Don't ask me every time it won't compile; it's usually just a missing brace or semi-colon. Try to look at your code *before* compiling it and see if it will give you an error. The more comfortable you get looking at code, the better a programmer you will become.

Think things through! Just because this class may not grade you harshly does not mean it is easy. In fact, one could argue that this is the hardest class they are taking because it requires you to think more than any other class you are in. Computer Programming is all about application and analyzation! In other classes, you memorize information which you must later recall in its exact form and you get full credit on a test and you rejoice and click your heels, and your parents give you money because you are so darn smart.

IN MY CLASS, THAT ISN'T IN THE CASE.

In this class, you don't need to memorize everything – you have all the resources in the world at your fingertips – but you need to know how to think and use the information. It is difficult. You will get frustrated. At times, you may have no idea what you are doing. But you are learning more about how to think and process information than any other class.

IF IT WERE EASY, EVERY ONE WOULD BE HERE.

And lastly, even if there isn't homework every night, it doesn't mean you shouldn't be programming on your own, coming up with your own projects, and doing your own research. Ultimately you will get out of this class what you put into it.

THE CHOICE IS YOURS.