**Comprehensive Level 1 Study Guide**

**Classes**

Classes are the foundation of a java program. Every Java program must have at least one class, although many programs use multiple classes.

class ExampleClass{

}

**Methods**

Methods are the life force of Java programs. Methods are a way to group code into a single procedure so that a task can be easily replicated. Every Java program has at least one method. Every Java program must also have a main method like this:

public static void main(String[] arg){

}

Every Java program begins in the main method. So if you write a java program and nothing happens when you run it, double check that you have a main method. This also means that no matter how much code is written before or above the main method, **the code *inside* the main method will alway run first.**

**example:**

class SampleClass{

int var1;

String var2;

double var3;

SampleClass(int var1, String var2, double var3){

this.var1 = var1;

this.var1 = var1;

this.var1 = var1;

}

void meaninglessMethod(){

System.out.println(“junk”);

}

public static void main(String[] arg){

System.out.println(“Hello!”); // <----- THIS CODE RUNS FIRST!!!

}

}

In the preceding example, even though the program has a bunch of code in it, we only need to look at the main method to tell what the program does. What is the result of running the program?

**Creating Methods**

All methods must be created inside of a class. There are four parts to a method and a method must have all four parts in order to be complete. They are:

1. **name** - How is the method identified? This is similar to naming a variable. The name should describe what the method does. For example, if the method averages two numbers, the the name can be something like **getAvarage** or **averageNumbers.**

2. **return type** - What information will the method give back? If the method does not return any information, then the word **void** is used. Otherwise, the return type will either be:

a primitive data type (int, boolean, double, float, char, etc…);

or a class name (String, JPanel, Robot, etc…).

3. **parameters** - What information does the method need to complete its task? This will be in the form of declaring variables in between a set of parentheses

e.g. **(**int x, String y**)**.

If no parameters are needed, then an empty set of parentheses **()** are used.

4. **definition** - The code that makes the method work. If the return type is NOT void, then you must have a return statement. A return statement is the word **return** followed by either a variable or a constant that matches the return type.

**examples:**

void doNothing(){

//nothing

}

int addNumbers(int x, int y){

int z = x + y;

return z;

}

double getPi(){

return 3.14159265359;

}

void printName(String name){

System.out.println(name);

}

String reverseWord(String word){

String reversedWord = "";

for(int i = word.length() - 1; i >= 0; i--){

reversedWord += word.charAt(i);

}

return reversedWord;

}

JFrame getSizedWindow(String title, int width, int height){

JFrame window = new JFrame(title);

window.setSize(width, height);

return window;

}

Notice that in all the preceding methods, if the return type is void, there is no return statement. And if the return type is not void, there is a return statement with the same data type as the return type.

**!!!!!!!!!!!!!!!YOU CAN NOT CREATE METHODS INSIDE OF METHODS!!!!!!!!!!!!!!!**

void function1(){

void function2(){

//THIS IS WRONG AND WILL RESULT IN AN ERROR

}

}

**Calling Methods**

In order to make all the code that your wrote inside your method to actually work, you need to call your method. To call a method, you need to be familiar with the parameters of the method. First, type the name of the method with parentheses **()** and determine if any data needs to go inside the parentheses.

If the method does not have any parameters (i.e. the parentheses are empty), then you can leave the parentheses empty when calling the method. So to call the **doNothing** method from the example above, it would be: **doNothing();**

If the method does have parameters (i.e. the parentheses are *not* empty), then you have to put data in the parentheses when calling the method. There needs to be either a variable or constant value that has the same data type as the parameters. When calling the method, they must also be listed in the same order as they are listed in the method definition (creation).

If the return type of the method is NOT void, then you will almost always need to save the value into a variable or print it directly to the screen. The variable must also be the same data type as the method's return type.

Methods MUST be called from another method.

//THIS IS CORRECT

class MyClass{

function1(){

function2(); <--calling function2 from function1

}

function2(){

}

}

//THIS IS WRONG

class MyClass{

function1(){

}

function2(); <--this is outside of a method and will result in an error

function 2(){

}

}

**calling methods from previous example:**

doNothing();

int sum = addNumbers(5, 7);

System.out.println("Pi is roughly equal to: " + getPi());

printName("Tidus");

String znarf = reverseWord("Franz");

JFrame coolWindow = getSizedWindow("hola", 500, 500);

**Constructors**

A constructor is a special type of method. There are two things that make constructors unique.

1. They do NOT have a return type.

2. Their name is the exact same as the class they are written in.

**example:**

class Eskimo{

void buildIgloo(){

//this is a normal method. It has a return type, and the name is not Eskimo

}

Eskimo(){

//this is a constructor. No return type and name is same as class.

}

}

**Creating Objects / Calling Constructors**

Constructors can not be called like any ordinary method. They can only be called by **creating an object.** Creating an object is like making a variable from a class. For example, let's create an object of the **Eskimo** class written above. First type the name of the class, and then type a name that you choose. This name can be just about anything you want, but it should be specific to the object that you are creating.

Eskimo aluki; <--an Eskimo variable named aluki

Currently, our **aluki** variable is valueless because we haven't set it equal to anything. So to validate it, or **initialize** it, we will call the Eskimo class constructor using the word **new.**

aluki = new Eskimo();

This can be, and is often done on one line.

Eskimo aluki = new Eskimo();

**more examples of creating objects:**

JPanel blankPanel = new JPanel();

Robot zippy = new Robot();

String emptyString = new String();

Object box = new Object();

**The Static Context**

At this stage, it is not yet important to know what the word **static** means. However, it is important to know that **static methods can only directly call other static methods.** Remember how every java program starts in the main method? Well, you may have also noticed that the main method has the word **static** in front of it.

public **static** void main(String[] arg){

}

So the main method can only directly call other static methods. But static methods (i.e. the main method) can call **constructors**. So that means we can create **objects** inside of our main method. This includes objects of the class that the main method is written in.

class SampleClass{

//constructor

SampleClass(){

System.out.println("Sample Class Created"); //<--this code runs third

}

//main method

public static void main(String[] arg){

System.out.println("Starting Program"); //<--this code runs first

SampleClass xyz = new SampleClass(); //<--this code runs second  
}

}

In static methods, once an object is created, that object can then be used to call methods from that class. This is done by typing the name of the object (The name you chose, not the name of the class.), then a period, then the name of the method you want to call (along with appropriate parameters).

**objectName.methodName(parameters);**

By doing this, you are getting out of the static context and are then free to call just about any methods you choose.

class SampleClass{

//constructor

SampleClass(){

System.out.println("Sample Class Created"); //<--this code runs third

}

//method that prints a word

void printWord(){

System.out.println("a word");

}

//main method

public static void main(String[] arg){

System.out.println("Starting Program"); //<--this code runs first

SampleClass xyz = new SampleClass(); //<--this code runs second

xyz.printWord(); //<--this code runs fourth  
}

}

What would be the output of the preceding program?