Review for test1

**What is method overloading and what is an advantage to using method overloading?**

Enables you to define the methods with the same name as long as their signatures are different. Ex: sort (int[ ]) sort(double[ ]) Same sort name

**What is method overriding?**

Overriding means to provide a new implementation for a method in the subclass.

**What are some differences between constructors and methods?**

Constructors are used to initialize the state of object using the *new* operator.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Name** | **Return type** | **invoke** |
| Methods | Any legal  name | any | call |
| Constructors | Same name  as class | none not  even void | new |

**What are 2 conditions for defining a constructor?**

A constructor must have as same name as class itself.

A constructor does not have a return type – not even void

**Give an example of a Wrapper class**

Wrapper class – processes primitive data type value as objects.

Ex: wrapping int into Integer class. Double into the Double class; char into Character class

**What is the difference between the keyword static and final? Provide an example of each.**

Static applies to the whole class

Final is constant, no sets, no changes.

Ex: apply delivery charge to all pizzas, we must define delivery\_charge as static.

**Can a subclass or child typically do more or less (in terms of methods) than its parent or superclass?**

Yes, because child has everything the parent has.

A subclass inherits accessible data fields and methods from its superclass and may also add new data fields and methods.

**Why would you need to use the instanceof operator?**

It’s used to compare, tests types of objects. -- It’s to check if an object is an instance of a specific class.

public void doSomething(Number param) {

if( param instanceof Double) {

System.out.println("param is a Double");

}

else if( param instanceof Integer) {

System.out.println("param is an Integer");

}

if( param instanceof Comparable) {

//subclasses of Number like Double etc. implement Comparable

//other subclasses might not -> you could pass Number instances that don't implement that interface

System.out.println("param is comparable");

}

}

Array must be a **double** or **int.**

**Extends vs implements**

**extends is for extending a class.**

**implements is for implementing an interface**

The difference between an interface and a regular class is that in an interface you can not implement any of the declared methods. Only the class that "implements" the interface can implement the methods.

public interface ExampleInterface{

public void do();

public String doThis(int number);

}

public class sub implements ExampleInterface{

public void do(){

//specify what must happen

}

public String doThis(int number){

//specfiy what must happen

}

}

**Superclass** defines common behavior for related subclasses. An interface can be used to define common behavior for cases (including unrelated classes).

**Abstract Class:** Cannot be used to create objects. An abstract class can contain abstract methods, which are implemented in concrete subclasses. -- Abstract classes are like regular classes, but you cannot create instances of abstract classes using the **new** operator. An abstract method if defined without implementation. Its implementation is provided by the subclasses.

An abstract method cannot be contained in a nonabstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined abstract. In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass.

**Instance Variables**

Instance variables are declared in a class, but outside a method, constructor or any block.

When a space is allocated for an object in the heap(pilha), a slot for each instance variable value is created.

Instance variables also:

Created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.

**Hold values that must be referenced by more than one method**, constructor or block, or essential parts of an object's state that must be present throughout the class.

Access modifiers can be given for instance variables.

**Visible for all methods**, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.

**Have default values**. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

Can be **accessed** directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. **ObjectReference.VariableName.**

**Class/Static Variables**

Class variables also known as static variables are:

Declared with the static keyword in a class, but outside a method, constructor or a block.

There would only be one copy of each class variable per class, regardless of how many objects are created from it.

Rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

Stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

Created when the program starts and destroyed when the program stops.

Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.

Can be **accessed** by calling with the class name **ClassName.VariableName.**

When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.