BCS 345 – Java Programming

Carlos Duque Rivera

Notes for Test 3: Exceptions, Inheritance, Polymorphism

Exceptions:

An exception is an indication of a problem that occurs when a program executes. There is no need to terminate the program when a problem happens, with exception handling we can allow the program to continue after dealing with the problem.

Normal error handling takes place in the “main line”, where you would have to check for errors even if they occur infrequently. With exception handling we can take the code that handles the exception out of the “main line”. This makes the code more readable and modifiable; it also allows you to choose which exceptions to handle.

Instead of letting the program end when an exception is thrown, we can handle the exception and let the program continue. To handle exceptions, we put all the code that may throw an exception inside of a try/catch block. If there is no matching catch block for the exception that was thrown, this would be an uncaught exception. To avoid uncaught exceptions, we can put additional catch blocks in our code.

There are checked exceptions and unchecked exceptions. Checked exceptions derive from the Exception class and a method that throws a checked exception must be called only if the method call is inside of a try/catch block or if the method call is inside of another method that throws that exception. Unchecked exceptions are also derived from the Exception class but not directly, they are derived from the RuntimeException class that derives from the Exception class. A method that throws an unchecked exception has no restrictions for calling it.

Checked exceptions are usually caused by conditions that are out of the control of the program.

A try statement can catch both a base class exception and an exception derived from that base class. To do this, we can include a catch block for the derived exception, which is the one being thrown, and include another catch block for the base class exception. We have to be careful with the order of the catch blocks when doing this.

Finally blocks are used for clean-up code, they run after try and catch are run. The finally block runs if:

* The try block exits by reaching the ending curly brace
* The try block exits using return, break or continue
* An exception is thrown and caught by the catch block
* An exception is thrown but not caught

The finally block does not run if the JVM is terminated ( System.exit(0) ).

Inheritance:

finalize() is a method that gets called when an object is actually garbage collected. There is no certainty of when will the garbage collector reclaim an area of memory, so we don’t know exactly when will the finalize() method of an object execute.

Inheritance is basically creating a new class from an existing class. The base class is the “super” class and the derived class is the “sub” class. The sub class has access to the variables from the super class if they are declared as “protected”.

To call the constructor of the base class inside of the derived class, we can use the “super()” keyword. Call to super must be the first statement in the constructor. If “super()” is not used in the default constructor of a derived class, Java will implicitly call the default constructor of the Base class. If there is no default constructor, Java creates one. If there is no default constructor but there is another constructor, Java will not create a default constructor and the program will crash if there is not a call to “super()”.

You can prevent a class from being inherited from by declaring it as final. In the other hand, we can make classes whose only purpose is to be inherited from. To do this, we can declare the class as abstract. We cannot make instances of abstract classes.

We can assign an instance of a derived class to an instance of a base class, but we cannot assign an instance of a base class into an instance of a derived class.

Java collection classes are all defined to accept objects. Any reference type (even user defined) can be stored in collections like Lists or Vectors.

Polymorphism & Interfaces

Polymorphism means “many forms” in Greek.

Override and Overload are examples of polymorphism in OOP.

The base class constructor cannot be overridden.

Interfaces define a set of behaviors and are implemented by classes. When a class implements an interface, the class must use the methods that are in the interface. You cannot call new in an interface but you can declare interface type variables.

Interfaces specify behaviors but they don’t specify the implementation of the methods, the implementation of the methods is defined in the classes that implement the interface. There is no limit to the number of interfaces that a class can implement.

Can only call methods on an interface reference that the interface has in its definition. The interface reference itself has to know the method exists (in interface definition) to be able to call it.

Classes are allowed to both derive from another class and implement an interface.