**Assigment 3**

Friday, August 24, 2018

1. Enlist differences between final, finally and finalize.

**final** – makes a variable constant, it cannot change its data, final methods cannot be overridden and a final class cannot be extended(inherited) and it is immutable.

**finally** – it is used in association with a **try/catch block** and guarantees a section of code will be executed even if an exception is thrown, it is mostly used for resource deallocation.

**finalize** – in java the garbage collector will always call this method just before the deletion/destroying an object, it also behaves like a resource deallocation.

1. Exception propagation – checked and unchecked exceptions.

- methods are executed in a stack. When an exception occurs in one method and it is not handled it drops down the stack until is handled. The list of methods is known as the **call stack** and the method of searching is **Exception Propagation.**

**- checked exceptions will have to be handled because it is a compile time error and the compiler will ask that the code to be handled in order to be compiled.**

**- unchecked exceptions don’t have to be handled, the program will be allowed to run but if the exception is met during runtime it will throw a runtime error and will crash the program.**

1. **Demonstrate custom exception.**

// A Class that represents use-defined expception

class MyException extends Exception {

    public MyException(String s)  {

         // Call constructor of parent Exception

         super(s);

      }

}

// A Class that uses above MyException

public class Main {

    // Driver Program

    public static void main(String args[])  {

        try {

            // Throw an object of user defined exception

            throw new MyException("Custom Exception");

        }  catch (MyException e)  {

             System.out.println("Caught");

             // Print the message from MyException object

             System.out.println(e.getMessage());

         }

  }

1. Exception handling rules in method overriding

An overriding method (the method of child class) can throw any unchecked exceptions, regardless of whether the overridden method (method of base class) throws exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw those checked exceptions, which have less scope than the exception(s) declared in the overridden method.

If parent class has not declared exception in the method, the child class can declare unchecked exceptions but cannot declare checked exceptions.

If the parent class has declared exception in the method, the child class can declare the same level or a subclass exception of the parent declared exception, or can declare no exception. It cannot declare an exception that is a super of the exception declared in the parent.

1. Exception classification

**checked exceptions** - a checked exception is an exception that occurs at the compile time, these are also called as **Compile Time Exceptions**. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

(ex. IOException)

**unchecked exceptions** - is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

(ex. ArithmeticException)

