Exceptions Summary

You must remember the following points:

1. The base class of all exceptions is **java.lang.Throwable**. **java.lang.Error** and **java.lang.Exception** are the only two subclasses of **Throwable**. **java.lang.RuntimeException** is a subclass of **Exception**.

2. **java.lang.Error** is used by the JVM to throw exceptions that have nothing to do with the program code as such but occur because of the environment. For example, java.lang.OutOfMemoryError. Error indicates serious problems that a reasonable application should not try to catch. Most such errors are abnormal conditions. Error and its subclasses are regarded as unchecked exceptions for the purposes of compile-time checking of exceptions.

3. **java.lang.Exception** is used by the programmer when it encounters exceptional situation in the program. Exception and its subclasses (except RuntimeExceptions) are called Checked Exceptions. Checked exceptions need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary. For example, java.io.IOException.

4. **java.lang.RuntimeException** extends Exception, which is used to report exceptional situations that cannot be predetermined at compile time. For example, IndexOutOfBoundsException or NullPointerException. RuntimeException and its subclasses are unchecked exceptions. Unchecked exceptions do not need to be declared in a method or constructor's throws clause.

5. A quick way to determine who should throw an exception is to see if the exception extends java.lang.Error. Errors are always thrown only by the JVM. Generally, RuntimeExceptions are also thrown by the JVM. However, it is ok for an application to throw a RuntimeException if it makes sense for the application to throw it in a given situation.

6. **Checked exceptions** that you should know about for the exam – java.lang.Exception, java.io.IOException extends java.lang.Exception, java.io.FileNotFoundException extends java.io.IOException.

7. **Unchecked exceptions** that you should about for the exam – All the ones mentioned below.

Exceptions thrown by JVM

1. **java.lang.ArrayIndexOutOfBoundsException extends java.lang. IndexOutOfBoundsException, which extends java.lang.RuntimeException**

Thrown when attempting to access an array with an invalid index value (either negative or beyond the length of the array).

Example :

```
int[] ia = new int[]{ 1, 2, 3}; // ia is of length 3.
System.out.println(ia[3]); //exception !!!
```

2. **java.lang.ClassCastException extends java.lang.RuntimeException**

Thrown when attempting to cast a reference variable to a type that fails the IS-A test.

Example :

```
Object s = "asdf";
StringBuffer sb = (StringBuffer) s; //exception at runtime because s is referring to a String.
```

3. **java.lang.NullPointerException extends java.lang.RuntimeException**

Thrown when attempting to call a method or field using a reference variable that is pointing to null.

Example :

```
String s = null;
System.out.println(s.length()); //NullPointerException at runtime because s is null.
```

4. **java.lang.ArithmeticException extends java.lang.RuntimeException**

Thrown when you try to divide by zero.

Example :

```
public class X {  int k = 0;
  public static void main(String[] args){
     k = 10/0; //throws java.lang.ArithmeticException
   }
}
```

5. **java.lang.AssertionError extends java.lang.Error**

Thrown to indicate that an assertion has failed i.e.when an assert statement's boolean test expression returns false. Note that the programmer does not explicitly throw AssertionError using the throw keyword. The JVM throws it automatically when an assertion fails.

Example:

```
private void internalMethod(int position)
{
      assert (position<100 && position >0) : position; //throws AssertionError if postion is > 100 or < 0
}
```

6. **java.lang.ExceptionInInitializerError extends java.lang.Error**

Thrown when any exception is thrown while initializing a static variable or a static block.

Example :

```
public class X {  int k = 0;
  static{
     k = 10/0; //throws java.lang.ArithmeticException but this is wrapped into a
                //ExceptionInInitializationError and thrown outside.
   }
```

```
}
```

7. java.lang.StackOverflowError extends java.lang.Error

Thrown when the stack is full. Usually thrown when a method calls itself and there is no boundary condition.

Example :

```
public void m1(int k){
  m1(k++);  // exception at runtime.
}
```

8. java.lang.NoClassDefFoundError extends Error

Thrown if the Java Virtual Machine or a ClassLoader instance tries to load in the definition of a class (as part of a normal method call or as part of creating a new instance using the new expression) and no definition of the class could be found. The searched-for class definition existed when the currently executing class was compiled, but the definition can no longer be found.

Example :

```
Object o = new com.abc.SomeClassThatIsNotAvailableInClassPathAtRunTime(); // exception at runtime.
```

Exceptions thrown by Application Programmer

As mentioned before, all instances of java.lang.Exception and its subclasses (except RuntimeExceptions) are generally thrown by the application programmer. In some cases, it is okay for the application programmer to throw RuntimeExceptions as well. The following are some important exception classes that you should remember for the exam.

1. java.lang.IllegalArgumentException extends RuntimeException

Thrown when a method receives an argument that the programmer has determined is not legal.

Example:

```
public void processData(byte[] data, int datatype)
{
      if(datatype != 1 || datatype != 2) throw new IllegalArgumentException();
      else …
}
```

2. java.lang.IllegalStateException extends java.lang.RuntimeException

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation. Note that this is different from IllegalMonitorStateException that is thrown by JVM when a thread performs an operation that it is not permitted to (say, calls notify(), without having the lock in the first place).

Example:

```
Connection c = …
public void useConnection()
{
      if(c.isClosed()) throw new IllegalStateException();
      else …
}
```

3. java.lang.NumberFormatException extends java.lang.IllegalArgumentException

It extends from IllegalArgumentException. It is thrown when a method that converts a String to a number receives a String that it cannot convert.

Example:

Integer.parseInt(“asdf”);

4. java.lang.SecurityException extends java.lang.RuntimeException

Thrown if the Security Manager does not permit the operation performed due to restrictions placed by the JVM. For example, when a java program runs in a sandbox (such as an applet) and it tries to use prohibited APIs such as File I/O, the security manager throws this exception. Since this exception is explicitly thrown using the new keyword by a security manager class, it can be considered to be thrown by application programmer.