Semester III

BCA311 JAVA Programming (Unit-5)

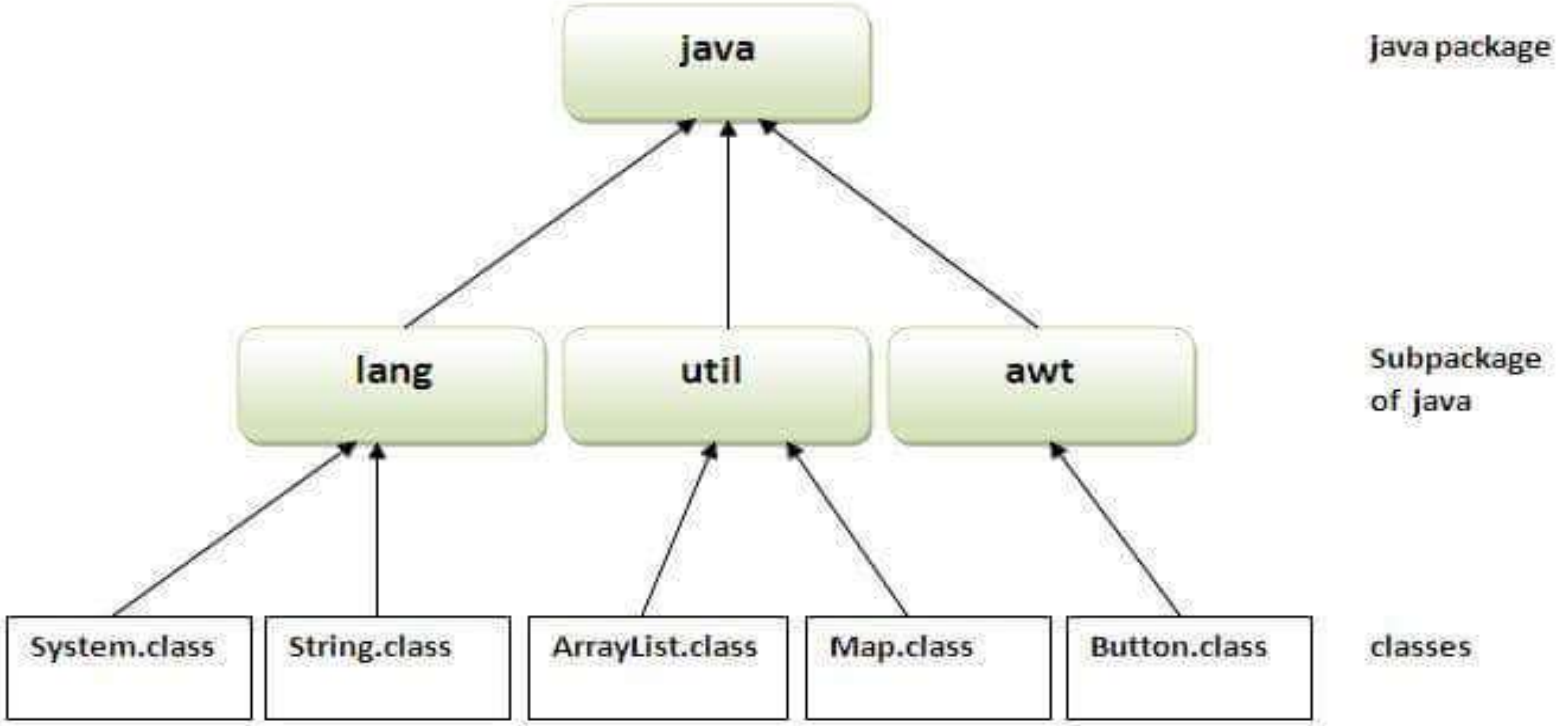
Harshita Mathur

Department of Computer Science

Email : hmathur85@gmail.com

Java Package

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.


java
lang
util
awt
System.class
String.class
ArrayList.class
Map.class
Button.class
Java package
Subpackage of java
classes

Advantage of Java Package

- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection.
- Java package removes naming collision.

Simple example of java package

- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to access package from another package?

There are three ways to access the package from outside the package.

- import package.*;
- import package.classname;
- fully qualified name.

Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

- The import keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
Output:Hello
```

Using fully qualified name

- If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

Output:Hello

Access protection in java packages

- In java, the package is a container of classes, sub-classes, interfaces, and sub-packages. The class acts as a container of data and methods. So, the access modifier decides the accessibility of class members across the different packages.

Access control for members of class and interface in java

Accessibility / Location / Access Specifier	Same Class	Same Package		Other Package	
		Child class	Non-child class	Child class	Non-child class
Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	Yes	No
Default	Yes	Yes	Yes	No	No
Private	Yes	No	No	No	No

Access protection in java packages

- The **public** members can be accessed everywhere.
- The **private** members can be accessed only inside the same class.
- The **protected** members are accessible to every child class (same package or other packages).
- The **default** members are accessible within the same package but not outside the package.

CLASSPATH setting for packages

- CLASSPATH is an environment variable which is used by Application ClassLoader to locate and load the .class files. The CLASSPATH defines the path, to find third-party and user-defined classes that are not extensions or part of Java platform. Include all the directories which contain .class files and JAR files when setting the CLASSPATH.

- You need to set the CLASSPATH if:

- You need to load a class that is not present in the current directory or any sub-directories.

CLASSPATH setting for packages

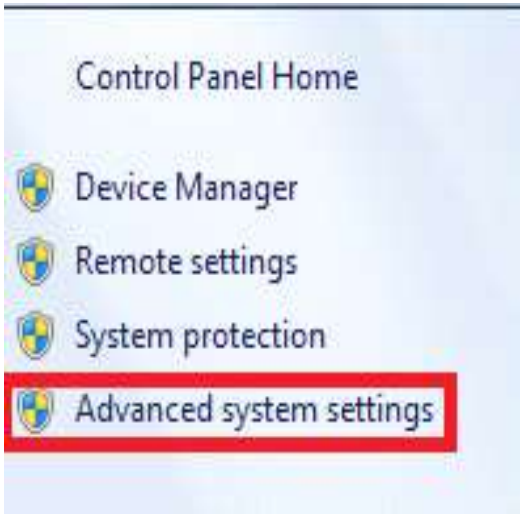
- The default value of CLASSPATH is a dot (.). It means the only current directory searched. The default value of CLASSPATH overrides when you set the CLASSPATH variable or using the -classpath command (for short -cp). Put a dot (.) in the new setting if you want to include the current directory in the search path.
- If CLASSPATH finds a class file which is present in the current directory, then it will load the class and use it, irrespective of the same name class presents in another directory which is also included in the CLASSPATH.
- If you want to set multiple classpaths, then you need to separate each CLASSPATH by a semicolon (;).

CLASSPATH setting for packages

- The third-party applications (MySQL and Oracle) that use the JVM can modify the CLASSPATH environment variable to include the libraries they use. The classes can be stored in directories or archives files. The classes of the Java platform are stored in rt.jar.
- There are two ways to ways to set CLASSPATH: through Command Prompt or by setting Environment Variable.
- Let's see how to set CLASSPATH of MySQL database:
- **Step 1:** Click on the Windows button and choose Control Panel. Select System.

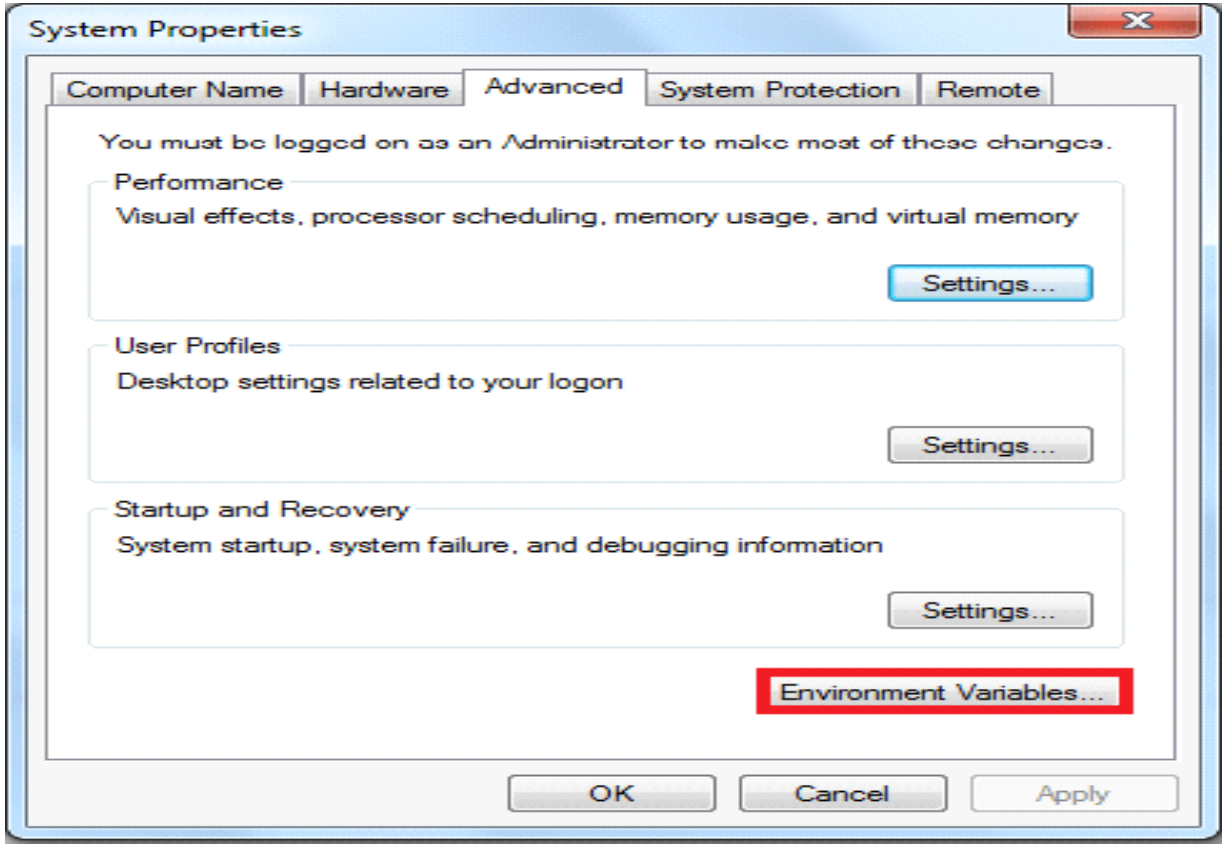
CLASSPATH setting for packages

- **Step 2:** Click on **Advanced System Settings**.



CLASSPATH setting for packages

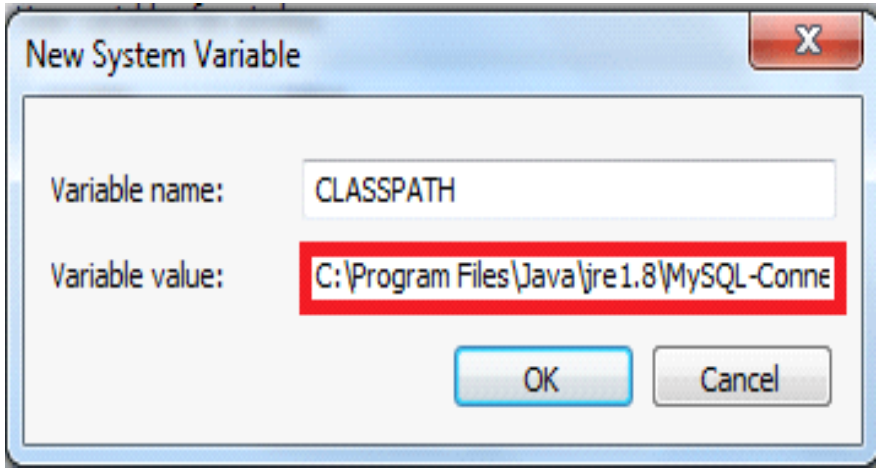
- **Step 3:** A dialog box will open. Click on Environment Variables.



CLASSPATH setting for packages

- **Step 4:** If the CLASSPATH already exists in System Variables, click on the Edit button then put a semicolon (;) at the end. Paste the Path of MySQL-Connector Java.jar file.
- If the CLASSPATH doesn't exist in System Variables, then click on the New button and type Variable name as CLASSPATH and Variable value as *C:\Program Files\Java\jre1.8\MySQL-Connector Java.jar;.;*
- Remember: Put ;.; at the end of the CLASSPATH.

CLASSPATH setting for packages



Set CLASSPATH in Windows Using Command Prompt

- Type the following command in your Command Prompt and press enter.

  set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\jre1.8\rt.jar;
- In the above command, The set is an internal DOS command that allows the user to change the variable value. CLASSPATH is a variable name. The variable enclosed in percentage sign (%) is an existing environment variable. The semicolon is a separator, and after the (;) there is the PATH of rt.jar file.

Naming Convention for packages

- Packages are usually defined
  using hierarchical naming pattern, with some levels in the hierarchy separated by periods (., pronounced "dot").
- Although packages lower in the naming hierarchy are often referred to as "subpackages" of the corresponding packages higher in the hierarchy, there is almost no semantic relationship between packages.
- The Java Language Specification establishes package naming conventions to avoid the possibility of two published packages having the same name.

Naming Convention for packages

- The naming conventions describe how to create unique package names, so that packages that are widely distributed will have unique namespaces.
- In general, a package name begins with the top level domain name of the organization and then the organization's domain and then any sub domains, listed in reverse order.
- The organization can then choose a specific name for its package. Subsequent components of the package name vary according to an organization's own internal naming conventions.

Naming Convention for packages

- For example, if an organization in Canada called MySoft creates a package to deal with fractions, naming the package ca.mysoft.fractions distinguishes the fractions package from another similar package created by another company.

- If a German company named MySoft also creates a fractions package, but names it de.mysoft.fractions, then the classes in these two packages are defined in a unique and separate namespace.

Exception Handling

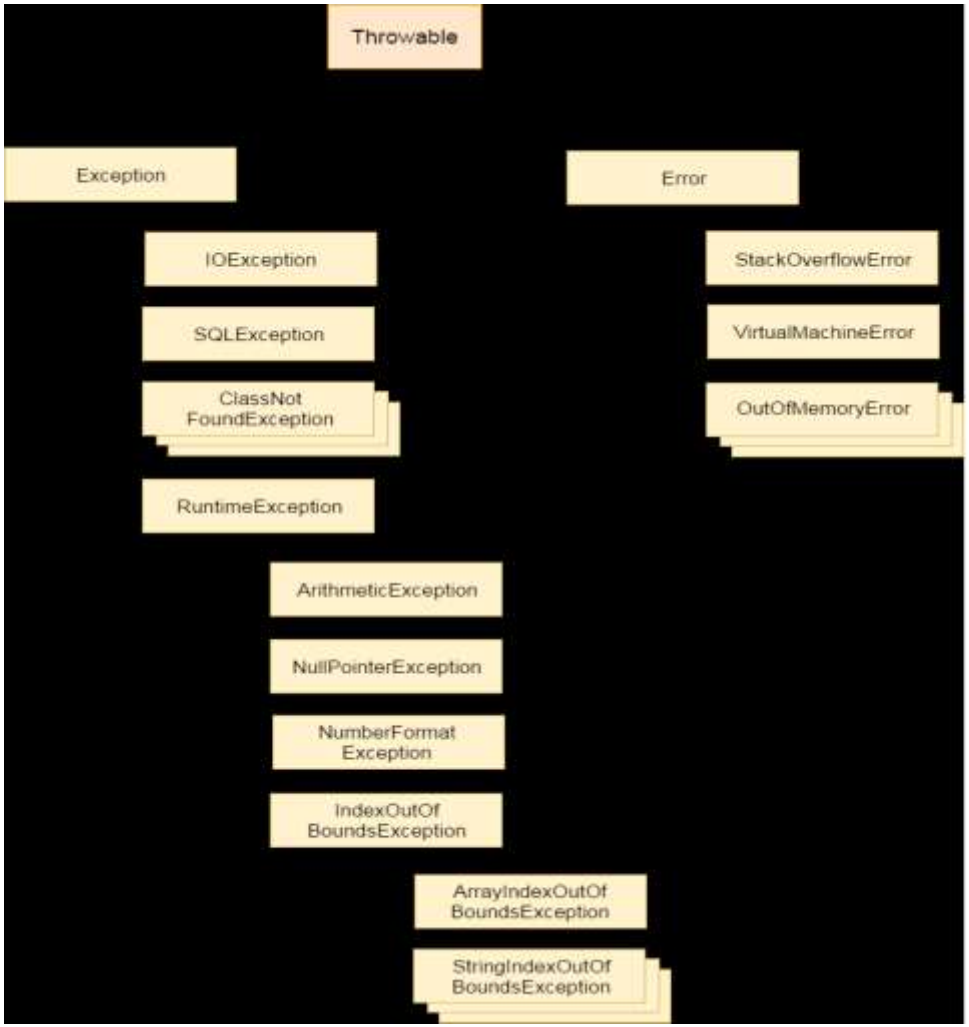
- The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- The core advantage of exception handling is to maintain the normal flow of the application.

statement 1;
statement 2;
statement 3;
statement 4;
statement 5;//exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;

- Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Hierarchy of Java Exception classes

- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException
StackOverflowError
VirtualMachineError
OutOfMemoryError

Java Exception Keywords

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

Keyword	Description
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
Throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
      //code that may raise exception
      int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

Common Scenarios of Java Exceptions

Multiple catch blocks

```
class JavaException
{
 public static void main(String args[])
 {
 try {
 int d = 1;
 int n = 20;
 int fraction = n / d;
 int g[] = { 1 };
 g[20] = 100;
}
 /*catch(Exception e)
{
System.out.println("In the catch block due to Exception = "+e);
}*/
```

Multiple catch blocks

```
catch (ArithmeticException e)
{
System.out.println("In the catch block due to Exception = " + e);
}
 catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("In the catch block due to Exception = " + e);
 }
 System.out.println("End Of Main");
}
}
```

Finally Block

- The finally block is **executed irrespective of an exception being raised** in the try block. It is **optional** to use with a try block.

```
try
{
statement(s)
 }
catch (ExceptiontType name)
{
statement(s)
}
finally
{
statement(s)
}
```

- In case, an exception is raised in the try block, finally block is executed after the catch block is executed.

Finally Block

```
class JavaException
{
 public static void main(String args[])
{
 try
{
 int d = 0; int n =20; int fraction = n/d;
}
catch(ArithmeticException e)
{
 System.out.println("In the catch block due to Exception = "+e);
 }
Finally
{
 System.out.println("Inside the finally block");
 }
 }
 }
```

Java throw keyword

- The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.

  Syntax: throw Instance

  Example: throw new ArithmeticException("/ by zero");

- Instance must be of type Throwable or a subclass of Throwable. For example Exception is a sub-class of Throwable and user defined exceptions typically extend Exception class.

java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1{
  static void validate(int age){
    if(age<18)
     throw new ArithmeticException("not valid");
    else
     System.out.println("welcome to vote");
  }
  public static void main(String args[]){
    validate(13);
     }
}
```

Output:

Exception in thread main java.lang.ArithmeticException:not valid

Java throws keyword

- The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- The throws keyword indicates what exception type may be thrown by a method.
- Syntax:

```
return_type method_name() throws exception_class_name{
//method code
}
```

Differences between throw and throws:

throw	throws
Used to throw an exception for a method	Used to indicate what exception type may be thrown by a method
Cannot throw multiple exceptions	Can declare multiple exceptions
Syntax: throw is followed by an object (new *type*)	Syntax: throws is followed by a class
used inside the method	used with the method signature

```
public class Main {
static void checkAge(int age) throws ArithmeticException {
if (age < 18)
 {
throw new ArithmeticException("Access denied - You must be at least 18 years old.");
 }
else
{
 System.out.println("Access granted - You are old enough!");
 }
}
 public static void main(String[] args)
 { checkAge(15); // Set age to 15 (which is below 18...)
}
}
```

Throwing your own Exception(Custom Exception)

- You can create your own exceptions in Java.
- All exceptions must be a child of Throwable.
- You just need to extend the predefined Exception class to create your own Exception.

class MyException extends Exception { }

Throwing your own Exception(Custom Exception)

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
class TestCustomException1{

   static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }
```

```
  public static void main(String args[]){
    try{
    validate(13);
    }catch(Exception m){System.out.println("Exception occured: "+m);}

    System.out.println("rest of the code...");
  }
}
```