Day 6: Package

- Package

Package

It is a collection of classes , interfaces and enums.

class: File
interface: File
enum: File
package: directory/folder

Benefits: Classification

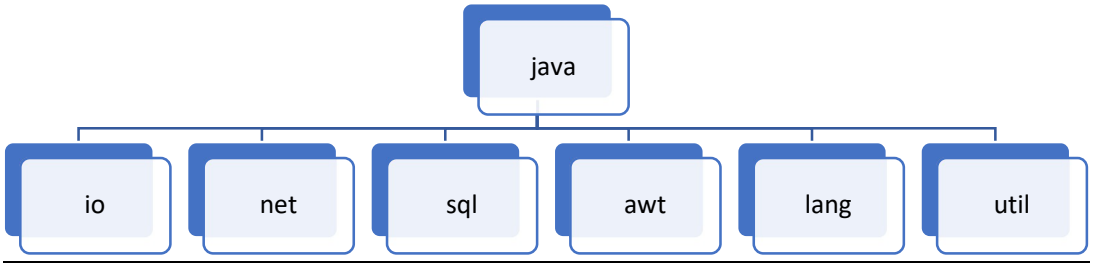
- Easier Naming
- Easier Access (Read/Write)

Types of Packages

1. Pre-Defined (Java API)
2. User-Defined

Note: Packages makes use of lower case

Pre-Defined (Java API)



To call classes or interfaces from a package "import" statement is used.

1. import <package>.*;

This will import all classes and interfaces from the specified package.

2. import <package>.<ClassName>;

This will import specified class from specified package.

3. import <package>.<InterfaceName>;

This will import specified interface from specified package.

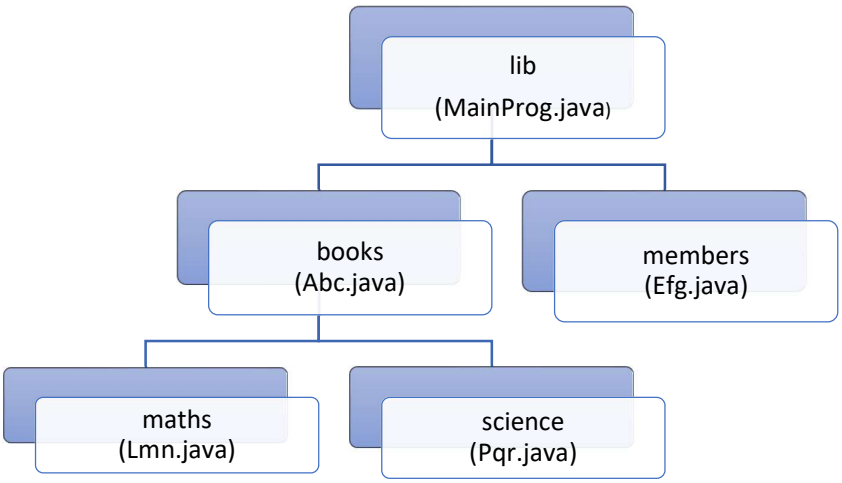
4. import <package>.<Enum>;

This will import specified interface from specified package.

NOTE:

- Whenever we import a package, it does not import the contents of its sub package
- By default contents of **java.lang** package is imported in every program. Classes from **java.lang** package include System, String, Wrappers Classes, Math, Object class.

Example: User Defined Packages



1. Create a folder called “lib”. Within “lib” folder create two sub-folders “books” and “members”, again within “books”, create two sub-folders “maths” and “science”.
2. Create 4 packaged classes Abc.java, Efg.java, Lmn.java

```
package lib.books;                                  //lib\books\Abc.java
public class Abc {
        public Abc() {
                System.out.println("This is Abc");
        }
}

package lib.books.maths;                            //lib\books\maths\Lmn.java
public class Lmn {
        public Lmn() {
                System.out.println("This is Lmn”);
        }
}
3. Create a class called "MainProg" within "lib"    // lib\MainProg.java
package lib;
import lib.books.*;
import lib.books.maths.Lmn;
import lib.books.science.Pqr;
import lib.members.Efg;
public class MainProg {
        public static void main(String args[])
                Abc a = new Abc();
                Efg b = new Efg();
                Lmn c = new Lmn();
                Pqr d = new Pqr();
        }
}
```

NOTE: Package classes cannot be compiled and interpreted from within the package

To compile and interpret a packaged class

Location: lib\MainProg.java (package lib)

```
D:\Data>javac lib\MainProg.java
```

D:\Data>java lib.MainProg

Location: lib\books\Abc.java (package lib.books)
D:\Data>javac lib\books\Abc.java
D:\Data>java lib.books.Abc

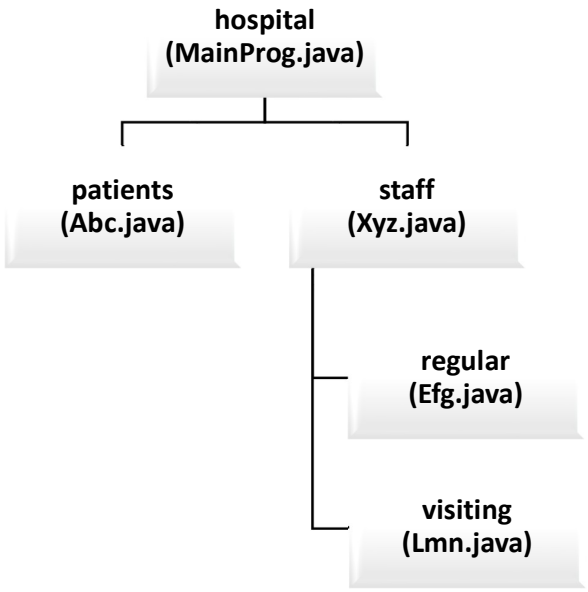
Assignment

Theory Questions

1. What are packages and why they are used? What are the various types of packages?
2. Give names of some of the pre-defined packages and which is the default package that gets imported in every program?
3. Why we use import statement? What are its variants?
4. How are packaged classes compiled and interpreted.

Practical Quesions

1. Create a class called **Abc** within a package called **mypack,** create another class called **Xyz** within the same package. Create main function within class **Xyz**, in which an object of **Abc** class is to be created. Compile and interpret it.
2. Create the following package hierarchy and perform the following operations:
    a. Write main function within **MainProg** that creates object of **Abc**, **Xyz**, **Efg**, and **Lmn**. Compile and Interpret it.
    b. Write main functions within **Xyz** that created objects of **Lmn** and **Efg**. Compiles and Interpret it.



Day-6: Exception Handling

- try
- catch
- finally
- throw
- throws

Types of Errors

1. Compile Time/Syntax Errors

Examples: missing ; , misspelling, missing ", using variable without declaration etc

2. Runtime Errors

Examples: division by Zero, Invalid Conversion, Accessing invalid index of an array.

- Runtime error results in an Exception
- Whenever Exception occurs Java creates an Object of "Exception" or its derive class
- Exception leads to abnormal termination of program giving some garbage on screen.
- Handling means making the user aware with problem
- For Exception Handling Java provides:
    - Constructs: try, catch, finally
    - Statements: throw, throws

try

- All statements that may result in an error are written within try.
- A method can have nested or multiple try blocks.

catch

- Every "try" should be followed by at least 1 catch block.
- Multiple catch blocks works like cases in switch only one will be executed depending on Exception generated.
- catch receives an argument of Exception or its derive class.
- catch is conditional

finally

- finally can be used as a substitute or along with catch.
- There can be only 1 finally.
- finally does not receive any argument.
- finally is unconditional.

throw

- It is used to generate an Exception

throws

- It is used to forward an Exception

Example:

```
class Exception1 {
        public static void main(String args[]) {
                int a = 10 , b  = 0, c = 0;
                c = a / b;
```

```
                System.out.println("Result is : " + c);
        }
}
```

Note: Division By Zero results in "***ArithmeticException***"

Example:

```
class Exception2 {
        public static void main(String args[]) {
                int a = 10 , b  = 0, c = 0;
                try{
                        c = a / b;
                }
                catch(ArithmeticException ae){
                        System.out.println("Divison by Zero");
                        System.out.println(ae.getMessage());
                        ae.printStackTrace();
                }
                System.out.println("Result is : " + c);
        }
}
```

Write a program that accepts a number as command line argument and display its square.

```
D:\Module1>java Exception3 5
Square : 25

D:\Module1>java Exception3 five
Whenever String to number conversion fails, java generates "NumberFormatException"

D:\Module1>java Exception3
Whenever we access invalid index of an array it results in "ArrayIndexOutOfBoundsException"
```

```
class Exception3 {
        public static void main(String args[]) {
                int num=0;
                try {
                        num = Integer.parseInt(args[0]);
                }
                catch(ArrayIndexOutOfBoundsException ae) {
                        System.out.println("Invalid Index");
                }
                catch(NumberFormatException ne) {
                        System.out.println("Invalid Format");
                }
                catch(Exception e) {
                        System.out.println("Some Error");
                }
                System.out.println("Square: " + Math.pow(num,2));
        }
```

```
}
```

Example:

```
class Exception4 {
	public static void main(String args[]) {
		int num=0;
		try {
			num = Integer.parseInt(args[0]);
		}
		finally {
			System.out.println("Square is : " + Math.pow(num,2));
		}
	}
}
```

Example:

```
class BigException extends Exception {		//BigException.java
	protected String msg;
	public BigException(){}
	public BigException(String msg) {
		this.msg = msg;
	}
	public String getMsg() {
		return msg;
	}
	public void setMsg(String msg) {
		this.msg = msg;
	}
}

class Exception5 {					//Exception5.java
	public static void main(String args[]) {
		int num=0;
		try {
			num = Integer.parseInt(args[0]);
			if(num>100) {
				BigException b = new BigException("Number too big");
				num=0;
				throw b;
			}
		}
		catch(NumberFormatException ne) {
			System.out.println("Invalid Format");
		}
		catch(ArrayIndexOutOfBoundsException ae) {
			System.out.println("Invalid Index");
		}
		catch(BigException be) {
			System.out.println(be.getMsg());
```

```
		}
		System.out.println("Square is : " + Math.pow(num,2));
	}
}
```

Example:

```
class Exception6 {
/*
	public static int convert(String s)  {
		int n = 0 ;
		try {
			n = Integer.parseInt(s);
		}
		catch(NumberFormatException ne) {
			System.out.println("INVALID FORMAT");
		}
		return n;
	} */


	public static int convert(String s) throws NumberFormatException {
		int n = 0 ;
		n = Integer.parseInt(s);
		return n;
	}

	public static void main(String args[]) {
		int num=0;
		try {
			num = Exception6.convert(args[0]);
		}
		catch(NumberFormatException ne) {
			System.out.println("Invalid Format");
		}
		catch(ArrayIndexOutOfBoundsException ae) {
			System.out.println("Invalid Index");
		}
		catch(Exception e) {
			e.printStackTrace();
		}
		System.out.println("Square is : " + Math.pow(num,2));
	}
}
```

Exceptions can be classified in 2 ways:

Classification 1:

1. Pre Defined Exceptions
2. User Defined Exceptions

Classification 2:

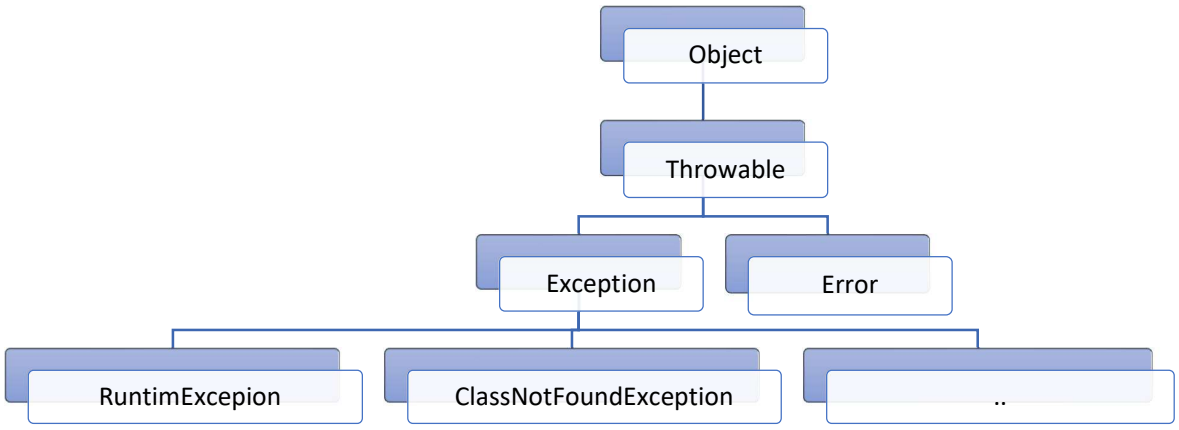
1. Unchecked Exceptions/Runtime Exceptions

- These exceptions are optional to be handled
- Generally, these except comes from java.lang package
- They are the derive classes of **RuntimeException**

2. Checked Exceptions

- These exceptions should be compulsorily handled
- These exceptions does not comes from java.lang package
- They are not the derive classes of **RuntimeException**

Hierarchy of Exception/Error



Derive classes of RuntimeException

- ArithmeticException
- IndexOutOfBoundsException
    - ArrayIndexOutOfBoundsException
    - StringIndexOutOfBoundsException
- ArrayStoreException
- ClassCastException
- IllegalArgumentException
    - NumberFormatException
    - IllegalThreadStateException
- IllegalMonitorStateException
- IllegalStateException
- NegativeArraySizeException
- NullPointerException
- SecurityException

Error

- LinkageError
    - NoClassDefFoundError
    - ClassFormatError
    - ..
- VirtualMachineError
    - InternalError
    - OutOfMemoryError
    - ..

Note: In case of multiple catch blocks, the sequence of catch should be from derive to base.

Assignment

Theory Questions

1. What is the need of Exception Handling? Explain the various statements and blocks that Java provides to handle exceptions.
2. What is the difference between throw and throws?
3. How can exceptions be classified? What is the difference between checked and unchecked excetions?

Practical Question

1. Write a program that accepts a number as command line argument and display its table. Handle all possible exceptions. If the number is greater than 25 it should generate a custom exception.
2. Create a static function that takes a number and return its cube, it the number is greater than 100 it should generate **NoCubeException.** Write a program that accepts a number as command line argument and display its cube. Cube should be calculated using the static function created.
3. In the above programs try to use finally