Java Packages

Built-in Packages

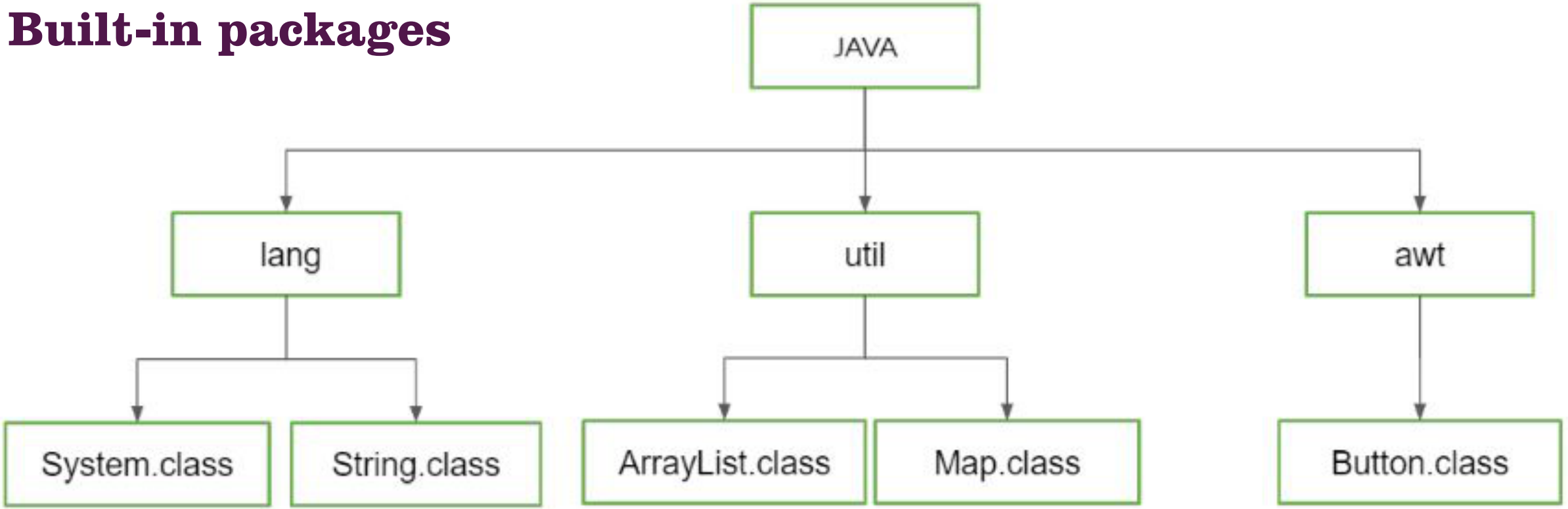
User-Defined Packages

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.

Built-in packages



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.

1.import package.*;
2.import package.classname;
3.fully qualified name.

1. 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.

1) Example of package that import the packagename.*

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

Output:Hello

```
//save by B.java
package mypack;
import pack.*;

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

2) Using packagename.classname

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

For file C.java:

```
// Save as C.java
package myutilities;
public class C {
  public void displayMessage() {
    System.out.println("Greetings!");
  }
}
```

Output:Hello

For file D.java:

```
// Save as D.java
package myapp;
import myutilities.C;
public class D {
  public static void main(String[] args) {
    C myObject = new C();
    myObject.displayMessage();
  }
}
```

3) 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.

```
//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

Subpackage in java

Package inside the package is called the subpackage. It should be created to categorize the package further.

```
package vehicles;

public class Vehicle {
    public void displayType() {
        System.out.println("I am a Vehicle");
    }
}
```

vehicles/cars/Car.java (with overridden method):

```
package vehicles.cars;
import vehicles.Vehicle;
public class Car extends Vehicle {
    @Override
    public void displayType() {
        System.out.println("I am a Car");
    }}
```

---

vehicles/trucks/Truck.java (with overridden method):

```
package vehicles.trucks;
import vehicles.Vehicle;
public class Truck extends Vehicle {
    @Override
    public void displayType() {
        System.out.println("I am a Truck");
    }}
```

TOPIC

Interface **in Java**

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{
   // declare constant fields
   // declare methods that abstract    by default.
}
```

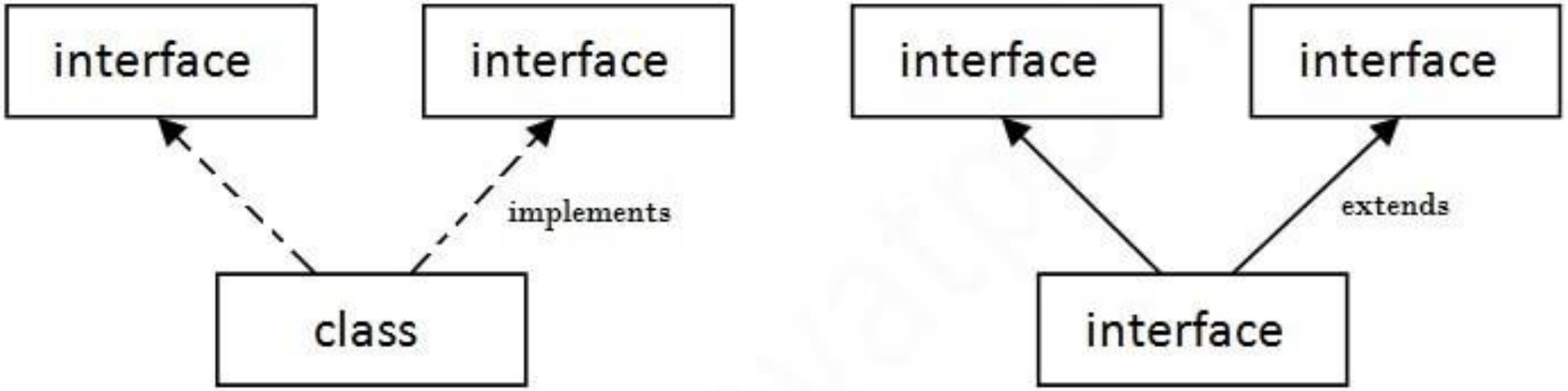
Example 1: Defining and Implementing an Interface

```
//Defining an Interface:

public interface Animal {
    void eat();
    void sleep();
}
//Implementing the Interface:
public class Dog implements Animal {
    @Override
    public void eat() {
        System.out.println("Dog is eating.");
    }
    @Override
    public void sleep() {
        System.out.println("Dog is sleeping.");
    }
}
public class TestAnimal {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        myDog.eat();
        myDog.sleep();
    }
}
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Multimedia Device Interface Implementation

This example demonstrates a class implementing two interfaces, Camera and MusicPlayer, allowing a device to have both functionalities.

```
interface Camera
 {
   void takePhoto();
}

interface MusicPlayer
 {
   void playMusic();
}
```

Implementation Class:

```
class SmartPhone implements Camera, MusicPlayer {
   public void takePhoto()
  {
      System.out.println("Photo taken.");
   }
   public void playMusic()
   {
      System.out.println("Music is playing.");
   }

   public static void main(String[] args) {
      SmartPhone myPhone = new SmartPhone();
      myPhone.takePhoto();
      myPhone.playMusic();
   }
}
```

Output:

```
Photo taken.
Music is playing.
```

Animal Behavior Interface Implementation

This example involves two interfaces, Walker and Swimmer, to represent the abilities of walking and swimming. A class Duck implements both to showcase how an animal can have multiple abilities inherited from multiple interfaces.

Interfaces:

```
interface Walker {
   void walk();
}
interface Swimmer {
   void swim();
}
```

Implementation Class:

```
class Duck implements Walker, Swimmer {
   public void walk() {
      System.out.println("Duck is walking.");
   }
   public void swim() {
      System.out.println("Duck is swimming.");
   }
   public static void main(String[] args) {
      Duck myDuck = new Duck();
      myDuck.walk();
      myDuck.swim();
   }
```

Output:

Duck is walking.

Duck is swimming.

Interfaces
Vs
Abstract classes

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.

6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

```
// Creating interface that has 4 methods
interface Shape {
   void draw(); // by default, public and abstract
   void erase();
   void move();
   void resize();
}
// Creating abstract class that provides the implementation of one method of Shape interface
abstract class AbstractShape implements Shape {
   public void move() {
      System.out.println("Moving the shape");
   }
}
// Creating subclass of abstract class, now we need to provide the implementation of the rest of the methods
class Circle extends AbstractShape {
   public void draw() {
      System.out.println("Drawing a circle");
   }
   public void erase() {
      System.out.println("Erasing the circle");
   }
   public void resize() {
      System.out.println("Resizing the circle");
   } }
// Creating a test class that calls the methods of Shape interface
class ShapeTest {
   public static void main(String args[]) {
      Shape shape = new Circle();
      shape.draw();
      shape.erase();
      shape.move();
      shape.resize();
```

```
Drawing a circle
Erasing the circle
Moving the shape
Resizing the circle
```