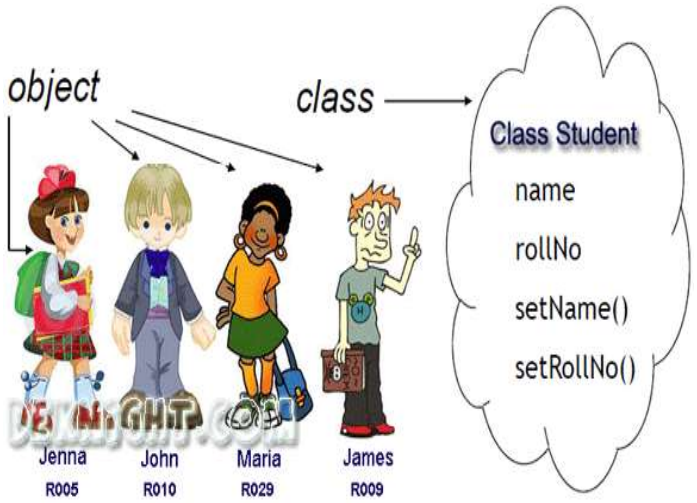
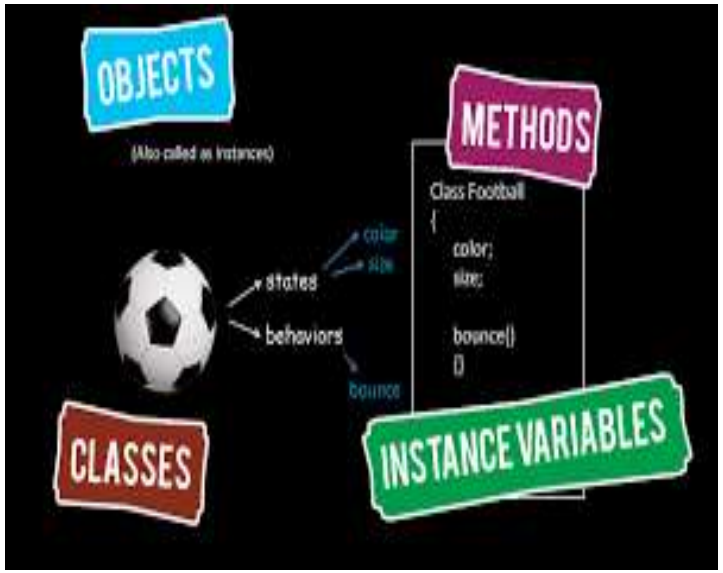
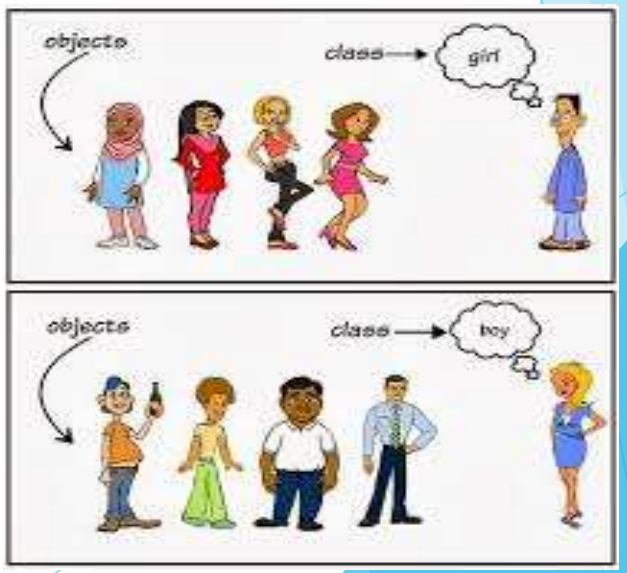
Classes, Objects and Methods



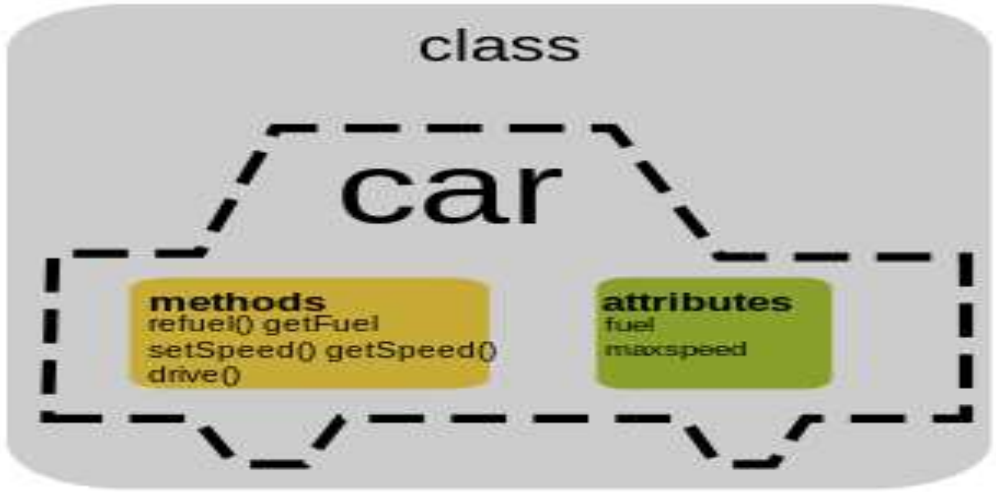




Class

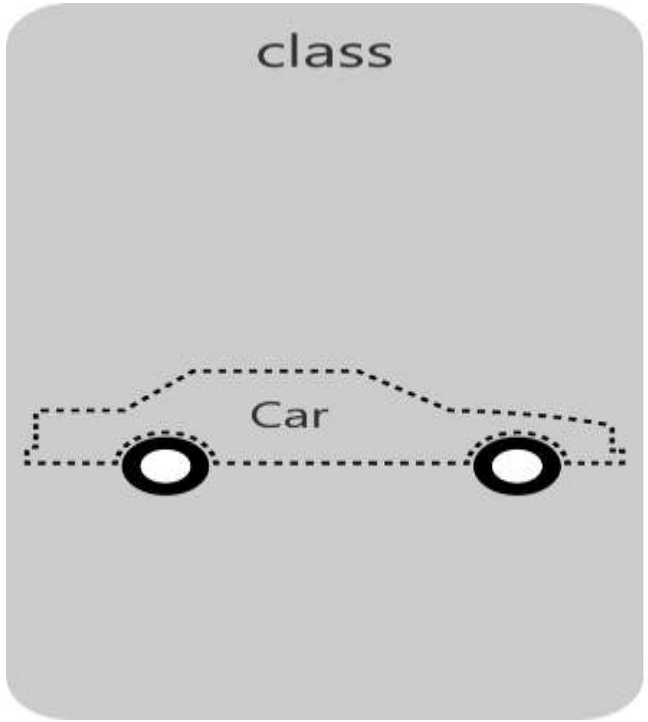
Structures are very similar to Classes in that they collect data together. A class is collection of objects of similar type. However, classes extend this idea and are made from two different things:

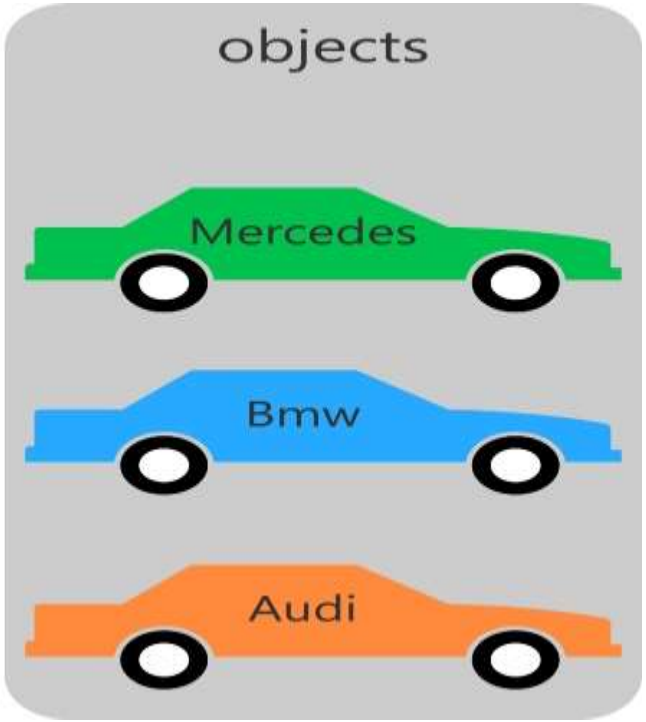
- **Attributes** - things that the object stores data in, generally variables.
- **Methods** - Functions and Procedures attached to an Object and allowing the object to perform actions



Object

- An **object** is a component of a program that knows how to perform certain actions and how to interact with other elements of the program.
- Objects are the basic units of object-oriented programming.





Defining a Class

Syntax :

```
class classname
{
    field declarations;
    method declaration;
 }
```

Ex:

```
class Rectangle
 {


  }
```

Fields Declaration

Syntax:

```
class classname
 {
    type variable_name;
 }
```

Ex:

```
class Rectangle
{
    int length;
 }
```

Methods declaration

Syntax:

type *methodname*(parameter-list)

{

method body;

}

Ex: void area(int a, int b)

{

method body;

}

Example of class

```
class Rectangle
{
    int length;
    int width;
    void getData(int x,int y)
    {
        length=x;
        width=y;
    }
}
```

Creating Objects

- Objects in java are created using the new operator.

- The new operator creates an object of the specified class and returns a reference to that object.

Ex:

```
Rectangle rect;                 //declare the object
rect=new Rectangle();    //instantiate the object
                or
Rectangle rect=new Rectangle();
```

Accessing class members

Syntax:

objectname.variablename=value;

objectname.methodname(parameter_list);

Ex:

```
rect.length=15;
rect.getData(15,10);
```

Example 1

```
class RectArea
{
 int length,width;
 void getData(int x,int y)
 {        length=x;
          width=y;
  }
  int Area()
  {
      int area=length*width;
      return(area);
    }
```

```
public static void main(String args[])
    {
        int a;
        RectArea rect=new RectArea();
        rect.getData(20,12);
        a=rect.Area();
        System.out.println("Area="+a);
     }
}
```

Example 2

```
class Rectangle
{
   int length,width;
   void getData(int x,int y)
   {
         length=x;
         width=y;
   }
   int rectArea()
   {
       int area=length*width;
       return(area);
    }
}
```

contd…

Contd…

```
class RectArea
{
    public static void main(String args[])
    {
        int a;
        Rectangle rect=new Rectangle();
        rect.getData(20,12);
        a=rect.rectArea();
        System.out.println("Area="+a);
     }
}
```

Question

WAP to find area of circle using classes, objects and methods.

Solution

```
class Circle
{
      double Area(int x)
   {
       double area=3.14*x*x;
       return(area);
    }
}
class CircleArea
{
     public static void main(String args[])
     {
           double area;
           Circle cir=new Circle();
           area=cir.Area(20);
           System.out.println("Area="+area);
      }
}
```

Scope of Variables

Classification of variables

Local variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method is entered and the variable will be destroyed once it exits the method.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method block.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Instance variable

- Instance variables are declared in a class, but outside a method.
- Instance variables hold values that must be referenced by more than one method.
- The instance variables are visible for all methods, constructors in the class.
- Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different class (when instance variables are given accessibility) should be called using the fully qualified name i.e. *ObjectReference.VariableName*.

Class variables

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.

- There would only be one copy of each class variable per class, regardless of how many objects are created from it.

- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.

Static Members

- The members that are declared static are called static members.
- These members are associated with the class itself rather than individual objects, the static variables and static methods are often referred to as class variables and class methods.
- Static variables are used when we want a variable common to all instance of class.
- Static variables and methods can be called without using objects. Instead they are called using class name.

e.g. static int count;

static int max(int x, int y)

Example of static variable

```
class Counter
{
   static int count=0;
   void count()
   {
     count=count+1;
     System.out.println(count);
   }
```

```
public static void main(String args[])
  {
      Counter c1=new Counter();
      c1.count();
      Counter c2=new Counter();
      c2.count();
      Counter c3=new Counter();
      c3.count();
   }
}
```

Example of static method

```
class Calculate
{
  static int cube(int x)
  {
        return x*x*x;
  }
 public static void main(String args[])
 {
   int result=cube(5);
  System.out.println(result);
  }
}
```

Method Overloading

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

Different ways to overload the method :

1. By changing number of arguments
2. By changing the data type
3. Sequence of Data type of parameters.

Example of Method Overloading by changing the no. of arguments

```
class DisplayOverloading
{
    void display(char c)
    {
        System.out.println(c);
    }
     void display(char c, int num)
    {
        System.out.println(c + " "+num);
    }
}
```

Contd…

Contd…

```
class Sample
{
   public static void main(String args[])
   {
      DisplayOverloading obj = new DisplayOverloading();
      obj.display('a');
      obj.display('a',10);
   }
}
```

Question

WAP to add different type of numbers using method overloading. Write different functions to add int, float etc.

Solution

```
class AddOverload
{
  void add(int x, int y)
  {
    int z=x+y;
    System.out.println("x+y="+z);
  }
  void add(float x, float y)
  {
    float z=x+y;
    System.out.println("x+y="+z);
  }
}
```

contd…

Contd…

```
class Addition
{
  public static void main(String args[])
  {
    AddOverload ad=new AddOverload();
    ad.add(20,10);
    ad.add(20.5f,10.5f);
   }
}
```

Different ways to overload the method :

1. By changing number of arguments
2. By changing the data type
3. Sequence of Data type of parameters.

```
import java.util.Scanner;
class add
{
   void cal(int a, int b)
   {
     int z=a+b;
     System.out.println(z);
   }
   double cal(int a, int b)
   {
    double z=a+b;
    return (z);
   }
```

```
public static void main(String args[])
  {
   add a1=new add();
   Scanner sc=new Scanner(System.in);
   int a=sc.nextInt();
   int b=sc.nextInt();
   a1.cal(a,b);
   double aa=a1.cal(a,b);
   System.out.println(aa);
  }
}
```

```
D:\oopm\19-20\Exp>javac add.java
add.java:9: error: method cal(int,int) is already defined in class add
   double cal(int a, int b)
          ^
add.java:21: error: incompatible types: void cannot be converted to double
    double aa=a1.cal(a,b);
                     ^
2 errors
```

```
import java.util.Scanner;
class add
{
   void cal(int a, int b)
   {
     int z=a+b;
     System.out.println(z);
   }
   void cal(double a, double b)
   {
    double z=a+b;
    System.out.println(z);
   }
```

```
 public static void main(String args[])
   {
    add a1=new add();
    Scanner sc=new Scanner(System.in);
    int a=sc.nextInt();
    int b=sc.nextInt();
    a1.cal(a,b);
    a1.cal(a,b);
   }
}
```

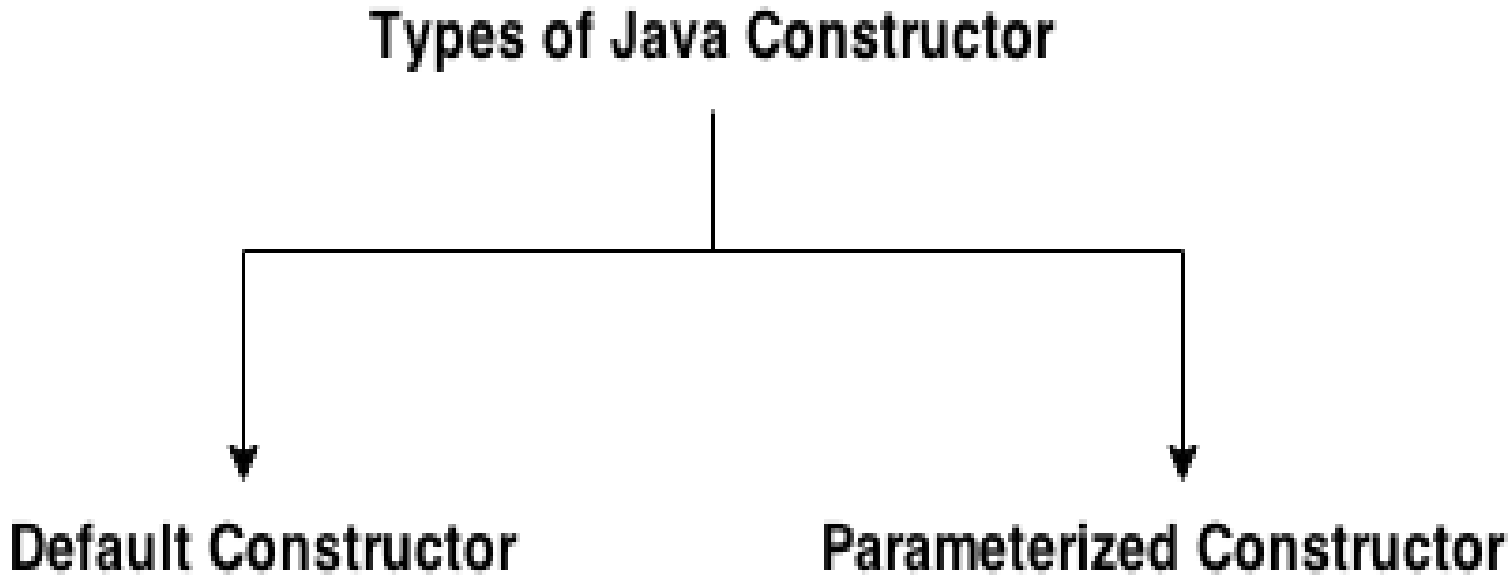
Constructor

- **Constructor** is a *special type of method* that is used to initialize the object.
- Java constructor is *invoked at the time of object creation*.
- It constructs the values i.e. provides data for the object that is why it is known as constructor.
- The java compiler provides a default constructor if you don't have any constructor.

Rules for creating java constructor

- Constructor name must be same as its class name
- Constructor must have no explicit return type

Types of constructors



Java Default Constructor

- A constructor that have no parameter is known as default constructor.
- Syntax of default constructor:

```
<class_name>()
{
}
```

Example

```
class Bike1
{
    Bike1()                 //default constructor
   {
        System.out.println("Bike is created");
    }
    public static void main(String args[])
    {
        Bike1 b=new Bike1();
    }
}
```

parameterized constructor

- A constructor that have parameters is known as parameterized constructor.
- Parameterized constructor is used to provide different values to the distinct objects.

Example

```
class Student
{
    int id;
    String name;
    Student(int i,String n)
    {
        id = i;
        name = n;
    }
    void display()
    {       System.out.println(id+" "+name);
    }
```

```
public static void main(String args[])
   {
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

```
111 Karan
222 Aryan
```

Question

WAP to read and display the details of book with the following specifications:

Data Members: book_name, book_author, book_price

Use Parameterized constructors to initialize data members of Book

Solution

```
class Book
{
    String name;
    String Author;
    int price;
    Book(String n,String a,int p)
    {
        name=n;
        Author=a;
        price=p;
     }
```

contd…

Contd…

```
void display()
    {
      System.out.println("name:"+name);
      System.out.println("Author:"+Author);
      System.out.println("Price:"+price);
    }
    public static void main(String args[])
   {
     Book b = new Book("Java:The Complete Reference","H.S.",300);
     b.display();
   }
}
```

Constructor overloading

- Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.

- The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example

```
class Student
{
    int id;
    String name;
    int age;
    Student(int i,String n)
   {
    id = i;
    name = n;
    }
    Student(int i,String n,int a)
    {
    id = i;
    name = n;
    age=a;
    }
```

contd...

Contd…

```
void display()
    {
       System.out.println(id+" "+name+" "+age);
     }
    public static void main(String args[])
   {
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

```
111 Karan 0
222 Aryan 25
```

Question

WAP to calculate area of circle and triangle using constructor overloading.

Solution

```
import java.util.Scanner;
class ConstructOverloadArea
{
    ConstructOverloadArea(int r)
    {
        float area=3.14f*r*r;
        System.out.println("Area of circle: "+area);
     }
    ConstructOverloadArea(int h,int b)
    {
        float area=0.5f*h*b;
        System.out.println("Area of triangle: "+area);
     }
```

contd…

contd…

```
public static void main(String args[])
  {
     int radius,height,base;
     Scanner sc=new Scanner(System.in);
     System.out.println("enter redius of circle, height and base of triangle");
     radius=sc.nextInt();
     height=sc.nextInt();
     base=sc.nextInt();
    ConstructOverloadArea s1 = new ConstructOverloadArea(radius);
    ConstructOverloadArea s2 = new ConstructOverloadArea(height,base);
  }
}
```

Passing and Returning Objects in Java

- Although Java is strictly pass by value, the precise effect differs between whether a primitive type or a reference type is passed.
- When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference.
- While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
- This effectively means that objects act as if they are passed to methods by use of call-by-reference.
- Changes to the object inside the method do reflect in the object used as an argument.

```
class Calculate
{
    int a;
     int b;
    Calculate (int x, int y)
    {       a = x;
            b = y;
    }
     void add(Calculate c1)
     {
         c1.a=c1.a +10;
          c1.b=c1.b +10;
       }
}
```

```
class CalculateDemo
 {
    public static void main(String args[])
    {
        Calculate c1 = new Calculate(10, 20);
        System.out.println("Before call:"+ c1.a +" "+c1.b);
        c1.add(c1);
        System.out.println("After call:"+ c1.a +" "+c1.b);
    }
}
```

```
D:\oopm\19-20\Exp>javac CalculateDemo.java

D:\oopm\19-20\Exp>java CalculateDemo
Before call:10 20
After call:20 30
```

Example

WAP to find area of rectangle by passing object as an arguments.

Solution

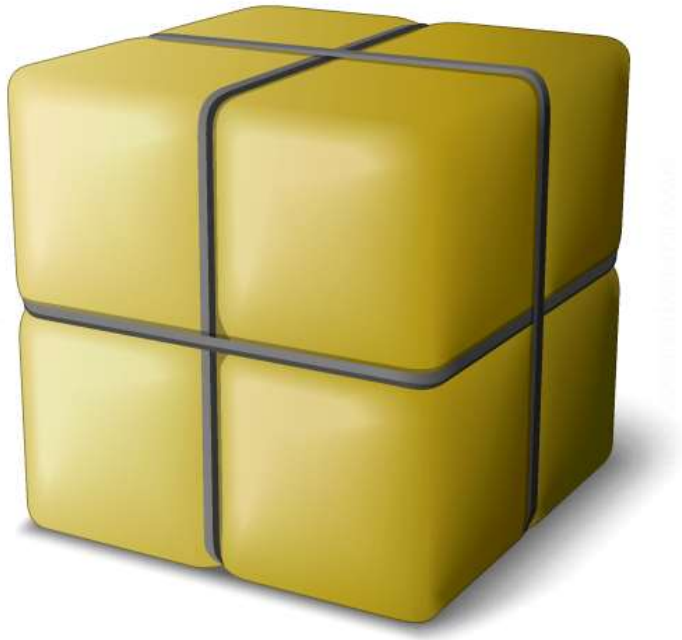
```
class Rectangle
{
    int length;
    int width;
    Rectangle(int l, int b)
    {       length = l;
            width = b;
    }
    void area(Rectangle r1)
   {
        int areaOfRectangle = r1.length * r1.width;
        System.out.println("Area of Rectangle : "  + areaOfRectangle);
    }
}
```

contd...

contd…

```
class RectangleDemo
 {
    public static void main(String args[])
    {
        Rectangle r1 = new Rectangle(10, 20);
        r1.area(r1);
    }
}
```

Packages : Putting classes together



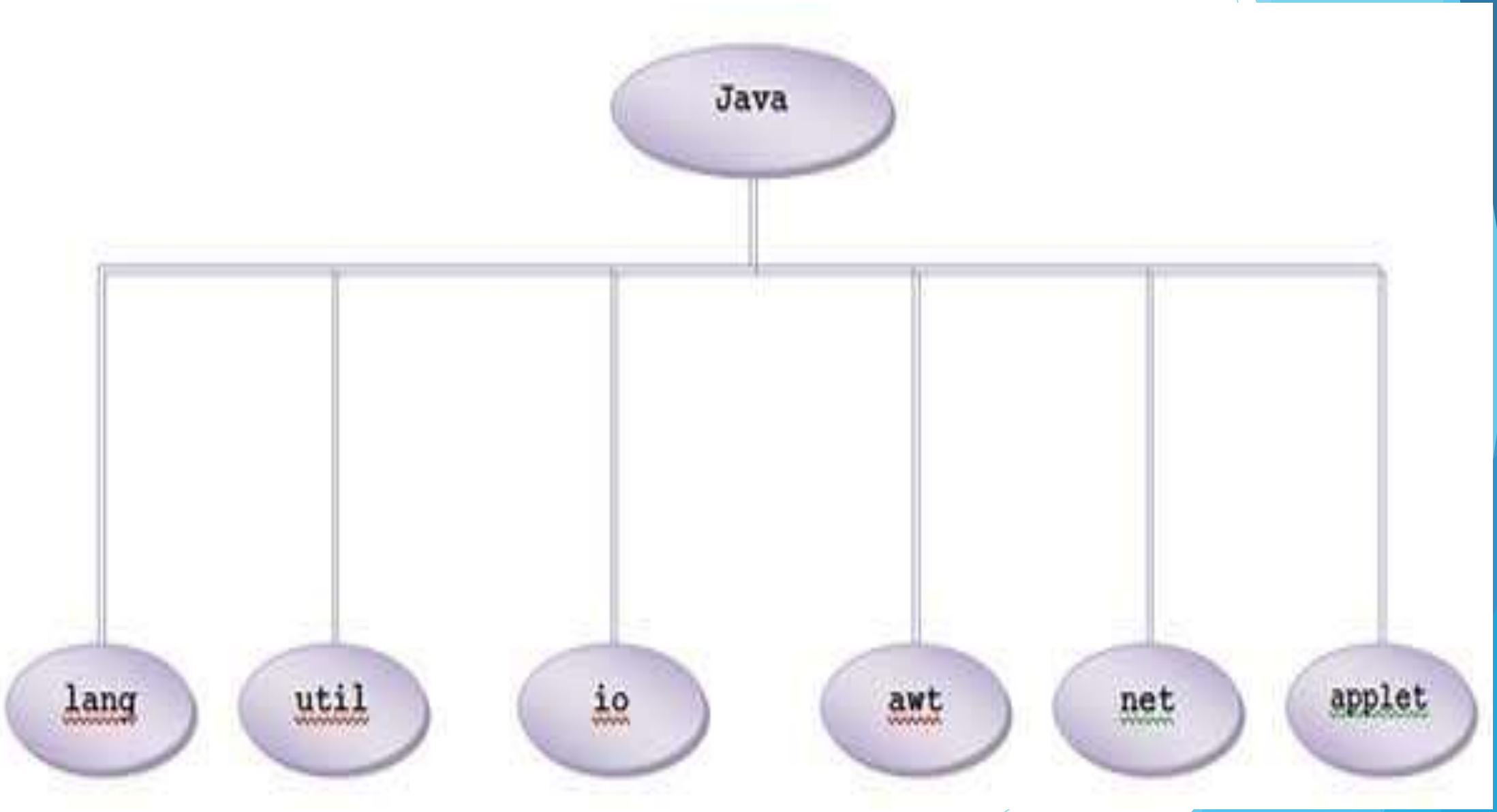
Introduction

- A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations etc.) providing access protection.

- Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations easier.

- Package in java can be categorized in two form:
  - built-in package/API packages (Application Programming Interface)
  - user-defined package.

Java API Packages



Using system Packages

- Syntax :

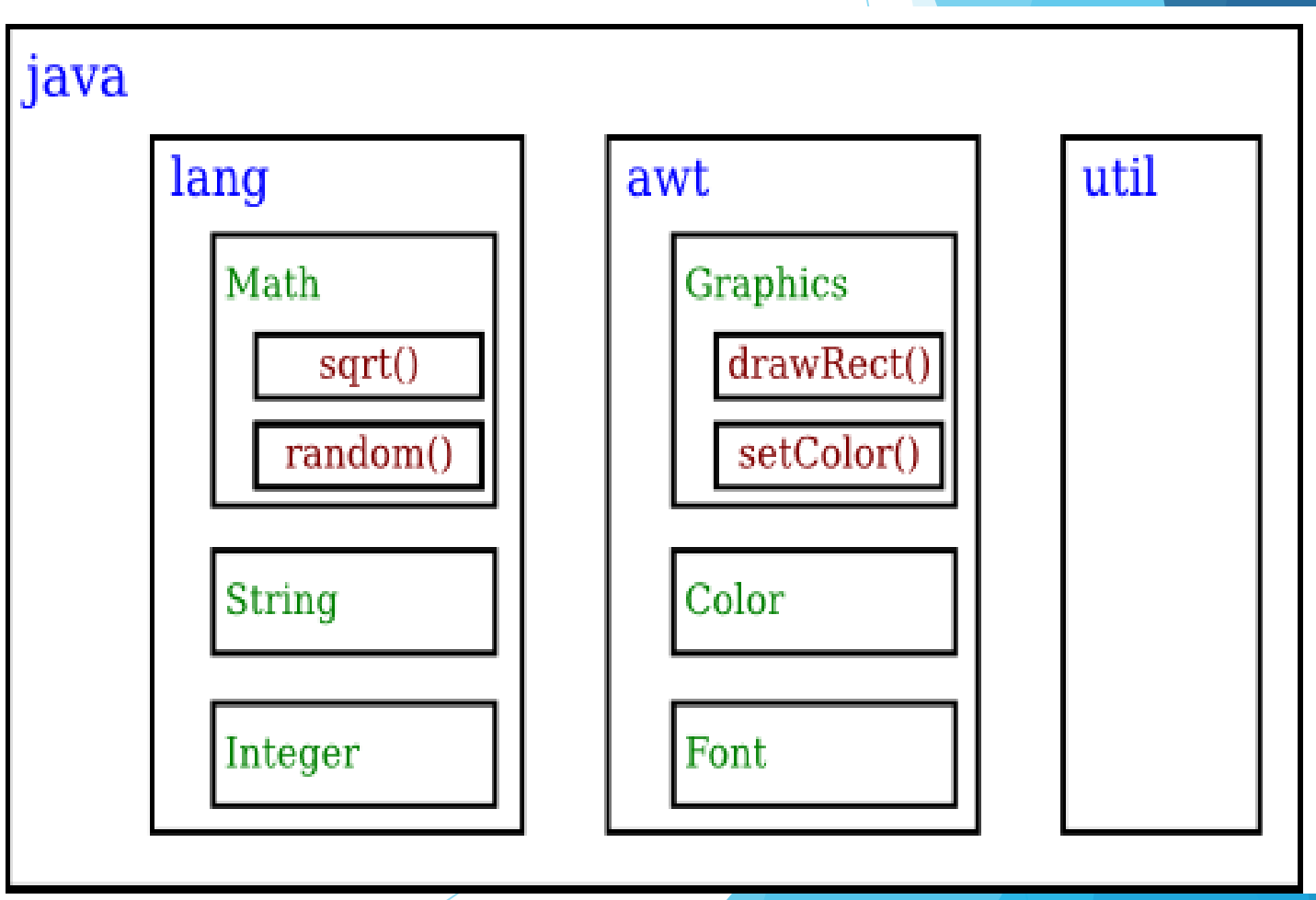
  import packagename.classname;

  or

  import packagename.*;

Example :

```
import java.util.scanner;
import java.util.*;
```



Creating User Defined Packages

Creating our own packages involves the following steps:

1. Declare the package at the beginning of the a file using the form.
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates *.class* file in the subdirectory.

Example

```
package firstpackage;
public class firstclass
{
    ..............
    body of class
   ..............
}
```

Accessing a Package

Syntax:

import package1[.package2][.package3].classname;

Example

import firstpackage.secondPackage.Myclass;

import packagename.*;

Adding a class to a package

- Define a class and make it public.
- Place the package statement before the class.

  package *packagename*;
- Store this as filename.*java* file under directory *packagename*.
- Compile the file. This will create .*class* file

Note: we can also add a non public class to a package using the same procedure.

Creating Package

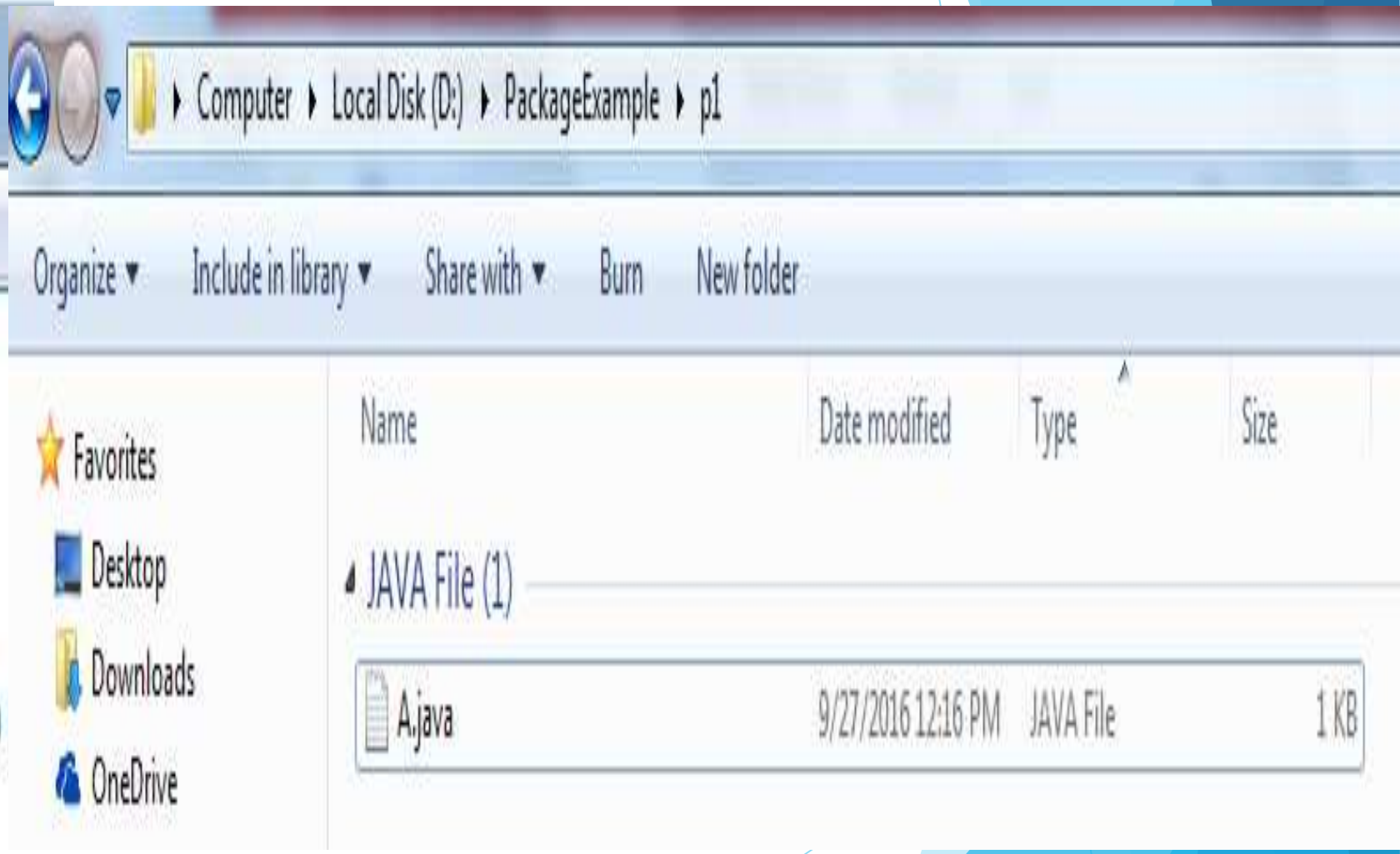
```
package p1;
public class A
{
  public void displayA()
   {
      System.out.println(“Class A");
   }
}
```

Creating Package

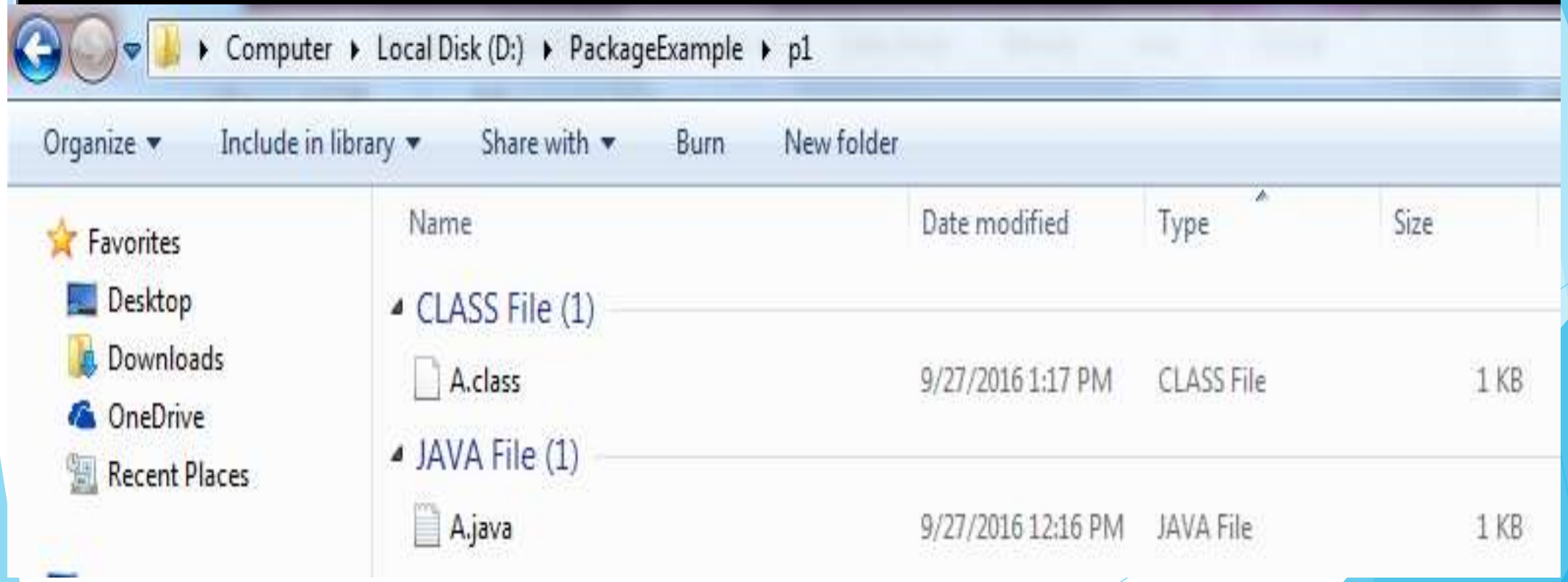
A.java - Notepad

File Edit Format View Help

```
package p1;
public class A
{
  public void displayA()
  {
    System.out.println("Class A")
  }
}
```



```
C:\Users\Pc-1>d:

D:\>cd D:\PackageExample\p1

D:\PackageExample\p1>set path=C:\Program Files\Java\jdk1.8.0_40\bin

D:\PackageExample\p1>javac A.java

D:\PackageExample\p1>
```



Importing a class

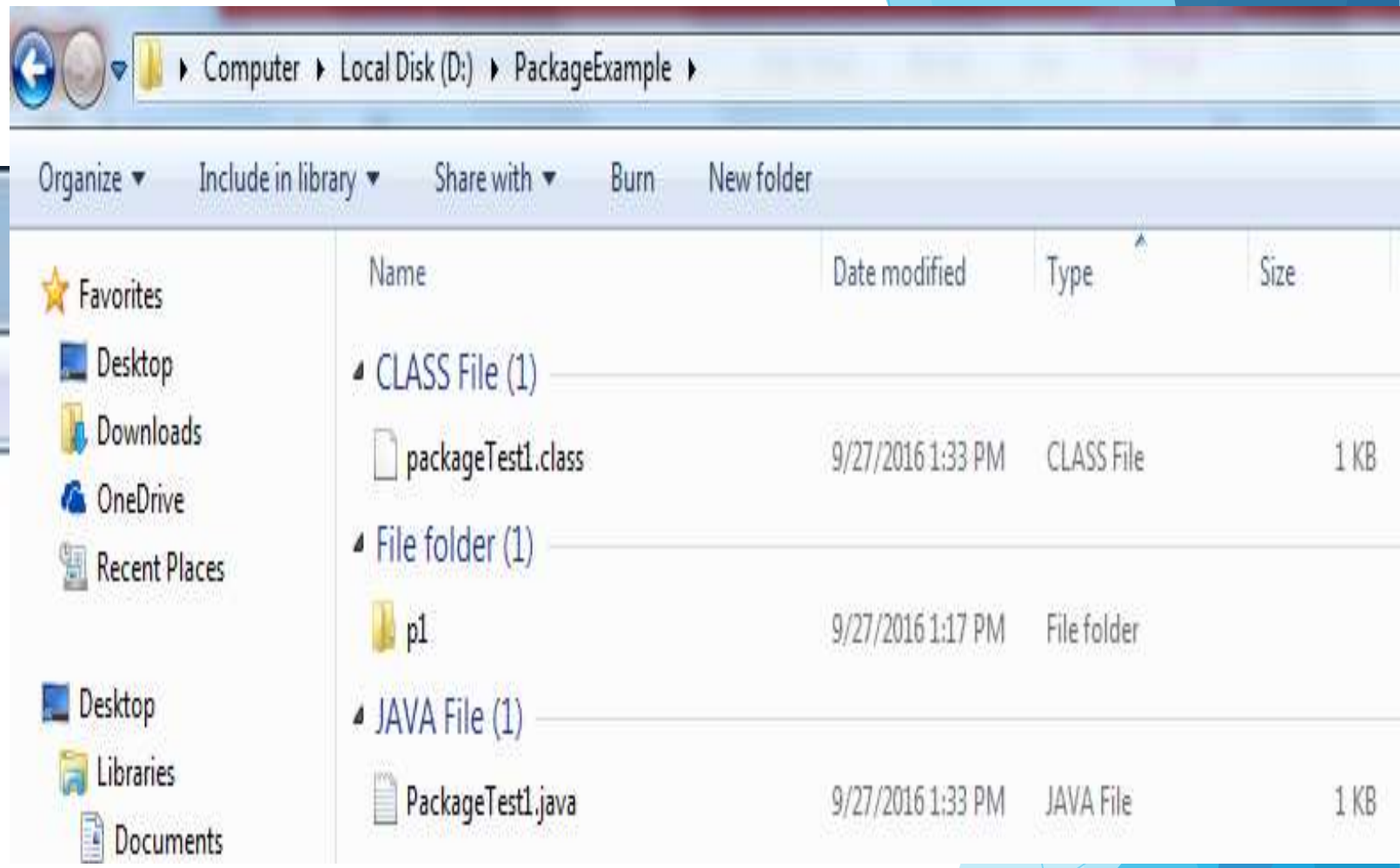
```
import p1.*;
class B
{
  public static void main(String args[])
  {
    A objA=new A();
     objA.displayA();
 }
}
```

Importing a class

PackageTest1.java - Notepad

File Edit Format View Help

```
import p1.A;
class packageTest1
{
  public static void main(String args[])
  {
    A objA=new A();
    objA.displayA();
  }
}
```



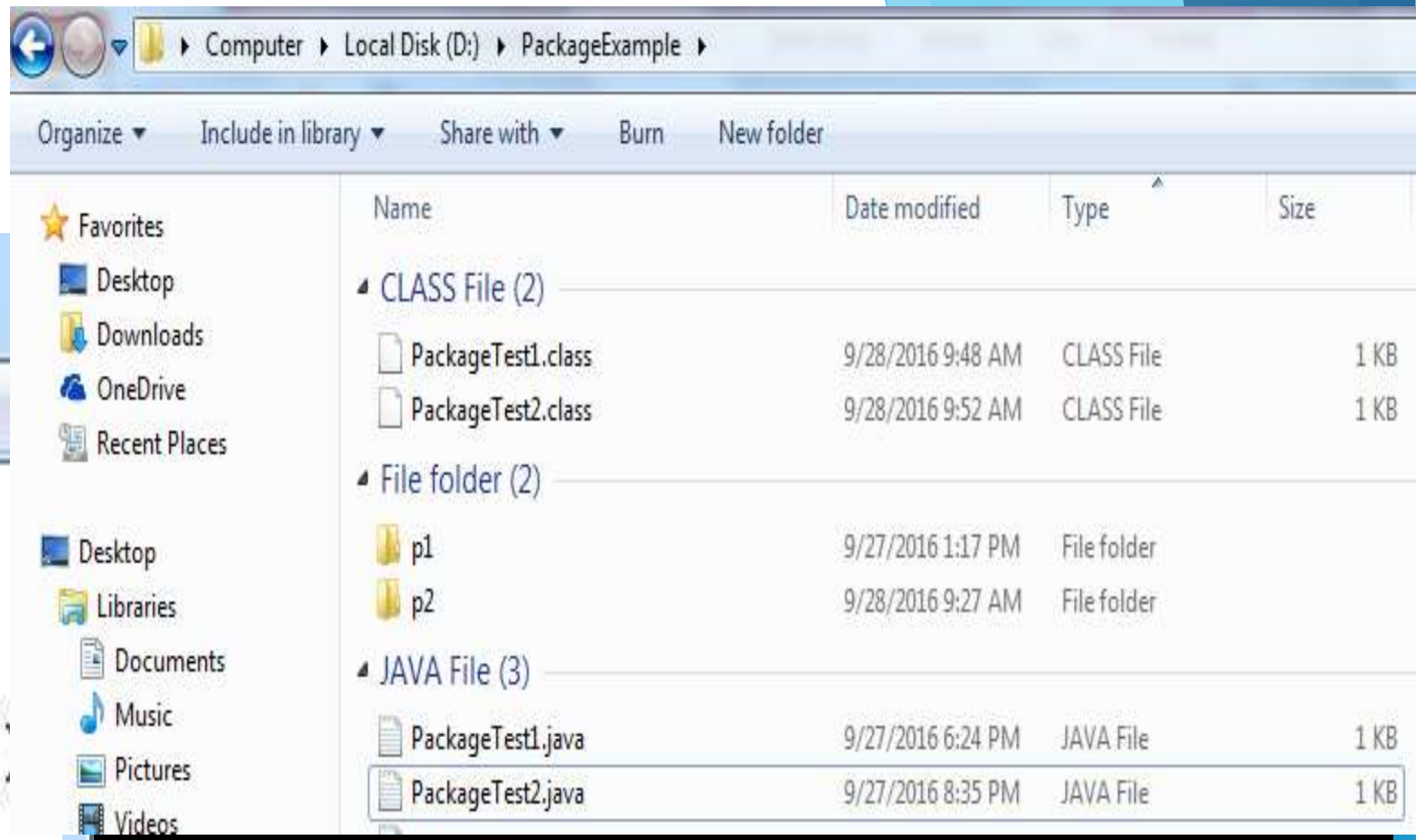

```
D:\PackageExample>javac PackageTest1.java

D:\PackageExample>java PackageTest1
Class A
```

Importing classes from other packages

PackageTest2.java - Notepad

File Edit Format View Help

```
import p1.A;
import p2.*;
class PackageTest2
{
  public static void main(String args[])
  {
    A obja=new A();
    B objb=new B();
    obja.displayA();
    objb.displayB();
  }
}
```




```
D:\PackageExample>javac PackageTest2.java

D:\PackageExample>java PackageTest2
Class A
Class B
m=10
```

Static import

- The static import statement can be used to import static members from classes and use them without qualifying the class name.
- Syntax:

```
import static packagename.subpackagename.classname.staticmembername;
```

or

```
import static packagename.subpackagename.classname.*;
```

Example without using static import

```
class Mathop
{
  public static void main(String args[])
  {
    int r=2;
    double area=java.lang.Math.PI*r*r;
    System.out.println("Area="+area);
   }
}
```

Example using static import

```
import static java.lang.Math.*;
class Mathop
{
  public static void main(String args[])
  {
    int r=2;
    double area=PI*r*r;
    System.out.println("Area="+area);
   }
}
```