Day-9 JA-111 Batch

Topics

- **Packages and Access modifiers, static import**
- **Abstract class**
- **Variable arguments in Java**
- **Enumerations in java**

Packages and Access modifiers

It is important for us to organize our Java files. In order to follow a modular approach, a set of related classes and interfaces can be grouped together at one place called package. Packages help in organizing our Java files. It is a mechanism of grouping similar types of classes and interfaces collectively based on functionality.

Creating packages and grouping related documents together mainly have the following advantages:

1. Modularization: Real-life Java projects will have several hundreds of classes and other files. Using packages provides a good structure for projects. It becomes easier to locate the related classes which improve efficiency.
2. Namespace Management: If there is a need for creating another class with the same name, then we can define two classes with the same name in different packages to avoid name collision.
3. Controlled Access: It provides access protection. Protected and default have package level access control. A protected member is accessible by classes in the same package and its sub-classes of the same or another package. A default member (without any access specifier) is accessible by classes in the same package only.
4. Data Encapsulation: They provide a way to hide classes, preventing other programs from accessing classes that are meant for internal use only.

Therefore, it is a good practice to use packages.

Packages can be divided into two categories:

1. **Built-in Packages**: Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit). They consist of a huge number of predefined classes and interfaces that are part of Java API. Some of the commonly used built-in packages are java.lang, java.io, java.util, etc.
2. **User-Defined Packages**: User-defined packages are those which are developed by users in order to group related classes, interfaces and sub-packages.

To create a package

1. File->New->Package; type package name
2. To make a file part of the package, the first line in the file will be-
   package package-name;

If a package statement is not used, then the Java files will be placed in the current default package. The naming convention for packages is to write the complete package in lowercase. Only public classes can be imported into different packages.

The import statement

The import statement is used to access clases and interfaces of a package in another package. It's general form is as follow-

```
import package-name.class-name;
```

The above syntax will import a specific class of package. if you want to import all classes and interfaces of a class at once then you need to use following syntax-

```
import package-name.*;
```

Tip: Importing all classes and interfaces of the package leads to increase in compile time of program because all imported file and classes need to compile. It does not have any impact on run time of program. Also importing a parent package does not import sub package implicitly i.e. sub package has to imported separately.

Another way to access classes and interfaces of a package in another package is by using the fully qualified name.

The Access Modifiers

	private	default	protected	public
Same class	Yes	Yes	Yes	Yes
Same package sub-class	No	Yes	Yes	Yes
Same package other class	No	Yes	Yes	Yes
other package sub class	No	No	Yes	Yes
other package sub class	No	No	No	Yes
Tip: default is also known as package private or friendly such that it is fully accessible in its package but not accessible outside the package protected = default + accessible in all subclass				

An Example

//folder: com/masai/p1 filename: A.java

```
package com.masai.p1;

public class A {
  private int i;
  int j;
  protected int k;
  public int l;

  public A() {
    System.out.println("Inside constructor of class com.masai.p1.A");
    i = 10;
    j = 20;
    k = 30;
    l = 40;
  }
}
```

//folder: com/masai/p1 filename: B.java

```
package com.masai.p1;

//same package sub-class
public class B extends A {
  B(){
    System.out.println("Inside constructor of class com.masai.p1.B");
    //System.out.println("i = " + i); CTE
    System.out.println("j = " + j);
    System.out.println("k = " + k);
```

```
    System.out.println("l = " + l);
  }
}
```

//folder: com/masai/p1 filename: C.java

```
package com.masai.p1;

//same package different class
public class C {
  C(){
    System.out.println("Inside constructor of class com.masai.p1.C");
    A a = new A();
    //System.out.println("a.i = " + a.i);  CTE
    System.out.println("a.j = " + a.j);
    System.out.println("a.k = " + a.k);
    System.out.println("a.l = " + a.l);
  }
}
```

//folder: com/masai/p1 filename: P1Main.java

```
package com.masai.p1;

//main class of package p1
public class P1Main {
  public static void main(String[] args) {
    System.out.println("Inside com.masai.p1.main");
    A a1 = new A();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    B b1 = new B();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    C c1 = new C();
  }
}
```

//folder: com/masai/p2 filename: A.java

```
package com.masai.p2;

public class A {
  public A(){
    System.out.println("Inside constructor of class com.masai.p2.A");
  }
}
```

//folder: com/masai/p2 filename: D.java

```
package com.masai.p2;
import com.masai.p1.A;

//other package sub class
public class D extends A {
  D(){
    System.out.println("Inside constructor of class com.masai.p2.D");
    //System.out.println("i = " + i);
    //System.out.println("j = " + j);
```

```
    System.out.println("k = " + k);
    System.out.println("l = " + l);
  }
}
```

//folder: com/masai/p2 filename: E.java

```
package com.masai.p2;
//other package other class
public class E {
  E(){
    System.out.println("Inside constructor of class com.masai.p2.E");
    //you can use fully qualified name also if import statement is not used
    com.masai.p1.A a1 = new com.masai.p1.A();
    //System.out.println("a1.i = " + a1.i);
    //System.out.println("a1.j = " + a1.j);
    //System.out.println("a1.k = " + a1.k);
    System.out.println("a1.l = " + a1.l);
  }
}
```

//folder: com/masai/p2 filename: P2Main.java

```
package com.masai.p2;
public class P2Main {
  public static void main(String[] args) {
    System.out.println("Inside com.masai.p2.main");
    A a2 = new A();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    D d2 = new D();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    E e2 = new E();
  }
}
```

//folder: com/masai/p3 filename: P3Main.java

```
package com.masai.p3;

public class P3Main {
  public static void main(String[] args) {
    System.out.println("Inside P3Main");
    com.masai.p1.A a1 = new com.masai.p1.A();
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    com.masai.p2.A a2 = new com.masai.p2.A();
  }
}
```

Output: Running P1Main.java

```
Inside com.masai.p1.main
Inside constructor of class com.masai.p1.A
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Inside constructor of class com.masai.p1.A
Inside constructor of class com.masai.p1.B
j = 20
k = 30
```

```
l = 40
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Inside constructor of class com.masai.p1.C
Inside constructor of class com.masai.p1.A
a.j = 20
a.k = 30
a.l = 40
```

Output: Running P2Main.java

```
Inside com.masai.p2.main
Inside constructor of class com.masai.p2.A
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Inside constructor of class com.masai.p1.A
Inside constructor of class com.masai.p2.D
k = 30
l = 40
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Inside constructor of class com.masai.p2.E
Inside constructor of class com.masai.p1.A
a1.l = 40
```

Output: Running P3Main.java

```
Inside P3Main
Inside constructor of class com.masai.p1.A
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Inside constructor of class com.masai.p2.A
```

The static import

Only static member(s) of a class can be imported instead of importing all members.

```
package com.masai.sprint3;

import java.util.Scanner;
import static java.lang.System.*;
import static java.lang.Math.sqrt;

class Demo{
  public static void main(String[] args) {
    Scanner sc = new Scanner(in);
    out.print("Enter number whose square root is to find ");
    double number = sc.nextDouble();
    out.println("The square root of " + number + "is " + sqrt(number));
    sc.close();
  }
}
```

Output

```
Enter number whose square root is to find 12321
The square root of 12321.0is 111.0
```

Advantage of static import:

Less coding is required if you have access to any static member of a class often.

The Disadvantage of static import:
If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Abstract classes

1. An abstract class is one that has abstract methods (not necessarily). To create an abstract class, we have to use abstract keyword before the class-name.

   ```
   abstract class class-name{
   ....
   }
   ```

   Tip: An abstract class can have variables, concrete methods and constructors also.

2. An abstract method is one that has only prototype but its body is missing. To create an abstract method, we have to use abstract keyword.

   ```
   abstract return-type method-name(para-list);
   ```

3. It is not possible to create an object of abstract class but it is perfectly okay to create reference variables.

4. For a subclass of abstract class, it is mandatory to provide definition of all abstract methods. If subclass fails to do the same then it must also be an abstract class.

5. A reference variable of abstract class can point to the object of its implementing class and when abstract method is called using that reference variable then calling will take place on the behalf of dynamic method dispatch so abstract class uses run time polymorphism

We Activity

Point out error, if any in the code below

```
package com.masai.sprint3;
class A{
  abstract void show();

  void foo(){
    System.out.println("Inside foo of class A");
  }
}

class B extends A{
  void fun(){
    System.out.println("Inside foo of class B");
  }
}

class C extends A{
  void show(){
    System.out.println("Inside show of class C");
  }

  void tez(){
    System.out.println("Inside tez of class C");
```

```
  }

  public static void main(String... args){
    A a = new A();
    A temp;
    temp = new C();
    temp.show();
  }
}
```

Answer:

```
//Solution: Either use abstract keyword with class A or write definition of
show() method in class A
package com.masai.sprint3;
abstract class A{
  abstract void show();

  void foo(){
    System.out.println("Inside foo of class A");
  }
}
//use abstract keyword with class B or write definition of show() method of
class A
abstract class B extends A{
  void fun(){
    System.out.println("Inside foo of class B");
  }
}

class C extends A{
  void show(){
    System.out.println("Inside show of class C");
  }

  void tez(){
    System.out.println("Inside tez of class C");
  }

  public static void main(String... args){
    //A a = new A(); Can;t create object of abstract class
    A temp;
    temp = new C();
    temp.show();
  }
}
```

An Example

```
package com.masai.sprint3;
abstract class Shape {
  private double dimOne;
  private double dimTwo;

  Shape(double dimOne, double dimTwo){
```

```
    this.dimOne = dimOne;
    this.dimTwo = dimTwo;
  }

  public double getDimOne() {
    return dimOne;
  }

  public void setDimOne(double dimOne) {
    this.dimOne = dimOne;
  }

  public double getDimTwo() {
    return dimTwo;
  }

  public void setDimTwo(double dimTwo) {
    this.dimTwo = dimTwo;
  }

  abstract double area();
  abstract double perimeter();
}
```

```
package com.masai.sprint3;
public class Triangle extends Shape{
  //The dimOne is for base and dimTwo is height
  public Triangle(double dimOne, double dimTwo) {
    super(dimOne, dimTwo);
  }

  @Override
  double area() {
    return 0.5 * getDimOne() * getDimTwo();
  }

  @Override
  double perimeter() {
    double hypotenuse = Math.sqrt(Math.pow(getDimOne(), 2) +
Math.pow(getDimTwo(), 2));
    double triPeri = Math.round((getDimOne() + getDimTwo() + hypotenuse) *
100)/100.0;
    return triPeri;
  }
}
```

```
package com.masai.sprint3;
public class Rhombus extends Shape {
  //The dimOne is for diagonal-1 and dimTwo is diagonal-2
  public Rhombus(double dimOne, double dimTwo) {
    super(dimOne, dimTwo);
  }
```

```
  @Override
  double area() {
    return 0.5 * getDimOne() * getDimTwo();
  }

  @Override
  double perimeter() {
    //code to find the side
    double side = Math.round(Math.sqrt(Math.pow(getDimOne()/2,2) +
Math.pow(getDimTwo()/2,2)) * 100)/100.0;
    //calculate and return perimeter
    return 4 * side;
  }
}

package com.masai.sprint3;
public class AbstractTester {
  public static void main(String[] args) {
    Shape shape;
    shape = new Triangle(7, 49);

    System.out.println("The area of triangle is " + shape.area());
    System.out.println("The perimeter of triangle is " + shape.perimeter());

    System.out.println();

    shape = new Rhombus(6, 8);

    System.out.println("The area of rhombus is " + shape.area());
    System.out.println("The perimeter of rhombus is " + shape.perimeter());
  }
}
```

Output:

```
The area of triangle is 171.5
The perimeter of triangle is 105.5

The area of rhombus is 24.0
The perimeter of rhombus is 20.0
```

Difference between abstract class and concrete class

- Concrete Classes are regular classes, where all methods are completely implemented but for abstract class some methods do not have functionality.
- Concrete Class can be instantiated but the abstract class cannot.
- Concrete Class can be final but the abstract class cannot be final.
- An abstract class is used to set common yet partially implement framework for the subclasses i.e. to use abstract class we need to create sub class of abstract class but for Concrete classes, no need to create subclass because they can be used directly.

Variable arguments in java

Whenever we wanted to pass an arbitrary number of arguments, we had to pass all arguments in an array or implement N methods (one for each additional parameter):

```
public int add(int a, int b)
public int add(int a, int b, int c)
public int add(int a, int b, int c, int d)
public int add(int a, int b, int c, int d, int e)
.....
public int add(int numbers[])
```

Better if we can write a single method that we can use for different number of arguments.
Solution: var-args

We can define them using a standard type declaration, followed by an ellipsis:

```
access-specifier return-type method-name(data-type... var-name) {
  // ...
}
```

Following points to be remember in case of the var-args-

- Using var-args a method can handle an arbitrary number of parameters (using an array internally).
- Each method can only have one var-args parameter
- The var-args argument must be the last parameter

An Example

```
package com.masai.sprint3;

class Demo{
  static int findMax(int... a) {
    int max = a[0];
    for(int i = 1; i < a.length; i++) {
      if(max < a[i])
        max = a[i];
    }
    return max;

  }
  public static void main(String[] args) {
    System.out.println(findMax(10, 15, 25, 5));
    System.out.println(findMax(20, 5));
  }
}
```

Output

```
25
25
```

The advantage of is that variable arguments help us avoid writing such boilerplate code

Enumeration

1. Using enums allows us to limit the selection within a set of values; this can help us to keep away from random input of user
2. Enumerations are group of named constants. All enums implicitly extend the `java.lang.Enum` class. A programmer can define constructors, methods and variables, inside enum.
3. The enum fields are implicitly static and final, and hence are constant during compile time. But they are instances of their enum type, constructed when the enum type is referenced for the first time. The

enum fields are written in capital letter according to java convention.

4. Enum can be written inside and outside class but not inside a method.
5. Enum variables can be used in an if statement or switch statement.
6. A static method called values() is automatically generated by the Java compiler for each enum. The values() method returns an array of all the constant values defined inside the enum. ordinal() method can be used to display values assigned to enum constants.

To create an enum

```
public enum enum_name { constant1, constant2, ..., constant n }
```

To create variable of class enum

```
access-modifier enum-name ref-name [ = enum-name.constant-name];
```

An Example

```
package com.masai.specifier;

public enum PizzaSize {
  SMALL, MEDIUM, LARGE
}

package com.masai.specifier;
import java.util.Scanner;
public class PizzaSizeTester {
  public static void main(String[] args) {
    System.out.println("All is well if well is all");
    for(PizzaSize pizza: PizzaSize.values()) {
      System.out.println("Name = " + pizza.name() + " Ordinal = " +
pizza.ordinal());
    }

    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=");
    Scanner sc = new Scanner(System.in);
    System.out.print("Please enter size [SMALL / MEDIUM / LARGE] ");
    String input = sc.next();
    //create an variable for enum
    PizzaSize pizza = PizzaSize.valueOf(input);

    //you can directly write value PizzaSize pizza = PizzaSize.SMALL;
    switch(pizza) {
      case SMALL:
        System.out.println("Good! You took it for change in taste");
        break;
      case MEDIUM:
        System.out.println("Ohh! Seems it your high tea");
        break;
      default:
        System.out.println("Great Choice for the lunch");
    }
  }
}
```

Output

```
All is well if well is all
Name = SMALL Ordinal = 0
Name = MEDIUM Ordinal = 1
Name = LARGE Ordinal = 2
-=-=-=-=-=-=-=-=-=-=-=-=-=-
Please enter size [SMALL / MEDIUM / LARGE] SMALL
Good! You took it for change in taste
```

The code

```
public enum PizzaSize {
  SMALL, MEDIUM, LARGE
}
```

is converted as

```
final class PizzaSize extends Enum{
  public static final PizzaSize SMALL = new PizzaSize();
  public static final PizzaSize MEDIUM = new PizzaSize();
  public static final PizzaSize LARGE = new PizzaSize();
}
```

You Activity

Say we have a student of science stream. Practical paper is of 30 marks and theory paper is of 70 marks each but for subject with no practical theory paper is of 70 marks.

The students of Science are further classified of science-maths and science-biology. The Registration prefix for student of science-maths and science-biology are "SM" and "SB" respectively i.e. registrationNumber of students will be SB1, SM2, SB3, SB4, SM4 and so on.... (SB1, SM1 is not possible i.e. counter after registration prefix is unique for every student and it is incremented for every student)

Complete the code of following classes Science, ScienceMathsStudent and ScienceBioStudent

```
abstract class Science{
  private static int counter;

  private String registrationNumber;
  private float physicsTheoryMarks;
  private float physicsPracticalMarks;
  private float chemistryTheoryMarks;
  private float chemistryPracticalMarks;

  static{
    counter = 0;
  }

  Science(String registrationPrefix, float physicsTheoryMarks, float
physicsPracticalMarks, float chemistryTheoryMarks, float
chemistryPracticalMarks){
    //write code to set instance variables
  }
```

```
  generate required setter and getter methods

  abstract public double getPercentage();
}

class ScienceMathsStudent extends Science{
  final static String MATHS_PREFIX = "SM";
  private float mathsMarks;

  ScienceMathsStudent(float mathsMarks, float physicsTheoryMarks, float
physicsPracticalMarks, float chemistryTheoryMarks, float
chemistryPracticalMarks){
    //write code to set instance variables
  }

  generate required setter and getter methods

  @Override
  public double getPercentage(){
    //write code here to return percentage up to two decimal places
  }
}

class ScienceBioStudent extends Science{
  final static String BIO_PREFIX = "SB";

  private float bioTheoryMarks;
  private float bioPracticalMarks;
  constructor, setter and getter methods

  ScienceBioStudent(float bioTheoryMarks, float bioPracticalMarks, float
physicsTheoryMarks, float physicsPracticalMarks, float chemistryTheoryMarks,
float chemistryPracticalMarks){
    //write code to set instance variables
  }

  @Override
  public double getPercentage(){
    //write code here to return percentage up to two decimal places
  }
}

class Tester{
  public static void main(String args[]){
    Science sc = null;
    sc = new ScienceMathsStudent(78, 55, 24, 58, 23);
    System.out.println("The total marks of student whose registration number "
+ sc.getRegistrationNumber() + " is " + sc.getPercentage());

    sc = new ScienceBioStudent(52, 23, 55, 24, 58, 23);
    System.out.println("The total marks of student whose registration number "
+ sc.getRegistrationNumber() + " is " + sc.getPercentage());
```

```
    sc = new ScienceBioStudent(49, 22, 54, 23, 59, 54);
    System.out.println("The total marks of student whose registration number "
+ sc.getRegistrationNumber() + " is " + sc.getPercentage());

    sc = new ScienceMathsStudent(83, 52, 23, 55, 26);
    System.out.println("The total marks of student whose registration number "
+ sc.getRegistrationNumber() + " is " + sc.getPercentage());
  }
}
```

Output

```
The total marks of student whose registration number SM1 is 68.33
The total marks of student whose registration number SB2 is 78.33
The total marks of student whose registration number SB3 is 87.00
The total marks of student whose registration number SM4 is 79.67
```