Q1] **What is an interface? How is an interface different from an abstract class in java? What are the access specifiers that can be used for data members in an interface and why?**

a) Interface:

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.

Java Interface also represents the IS-A relationship.

b)**How interface is different from an abstract class**

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.

c) **access specifiers that can be used in an interface:**

As of Java7 you can have only public, abstract as modifiers for the methods of an interface.
From Java8 onwards interfaces allow default methods and static methods.

Static methods – A static method is declared using the static keyword and it will be loaded into the memory along with the class. You can access static methods using class names without instantiation.You need to call the static method of an interface using the name of the interface.

```java
interface MyInterface{
  public void demo();
  public static void display() {
    System.out.println("This is a static method");
  }
}
public class InterfaceExample{
  public void demo() {
    System.out.println("This is the implementation of the demo method");
  }
  public static void main(String args[]) {
    InterfaceExample obj = new InterfaceExample();
    obj.demo();
    MyInterface.display();
  }

}
```

Output

```
This is the implementation of the demo method

This is a static method
```

Q2] **Write a Java program to create an interface Searchable with a method search() that searches an element in an array of integers. Create two classes LinearSearch and BinarySearch that implement the Searchable interface and provide their own implementations of the search() method**

```java
interface Searchable{
  public int search (int[] ar, int k);
}

class LinearSearch implements Searchable{
    public int search(int[] ar, int k){
        for(int i=0;i<ar.length;i++){
            if(ar[i] == k){
                return i;
            }
        }
        return -1;
    }
}

class BinarySearch implements Searchable{

    public static int binarySearch(int arr[], int first, int last, int key){
        int mid = (first + last)/2;
        while( first <= last ){
            if ( arr[mid] < key ){
                first = mid + 1;
            }
            else if ( arr[mid] == key ){
                return mid;
            }
            else{
                last = mid - 1;
            }
            mid = (first + last)/2;
        }
        if ( first > last ){
            return -1;
        }
        return 0;
    }

    public int search(int[] ar, int k){
        return binarySearch(ar,0,ar.length,k);
    }
}

public class Main
{
  public static void main (String[]args)
  {
    LinearSearch ls = new LinearSearch();
    BinarySearch bs = new BinarySearch();
    int[] arr={12,23,34,45,56,67,78,90};
    int a = ls.search(arr, 45);
    if(a==-1)
        System.out.println("Not found");
    else
        System.out.println("Found at "+(a+1)+" index.");
    int b = bs.search(arr, 90);
    if(b==-1)
        System.out.println("Not found");
    else
        System.out.println("Found at "+(b+1)+" index.");
  }
}
```

Output

```
Found at 4 index.
Found at 8 index.
```

Q3]**How does inheritance work in an interface? Why an Interface can extend more than one Interface but a Class cannot extend more than one class? Explain with an example code.**

a) Java does not support the concept of multiple inheritances to avoid the diamond problem encountered in C++ without using a virtual base class. However, Java supports multiple interface inheritance where an interface extends more than one super interface.

b)why an interface can extend more than one interface but a class can’t extend more than one class

The interface is a pure abstraction model and does not have inheritance hierarchy like classes. That's why interfaces are able to extend more than one interface but a class can't extend more than one class because Java doesn't allow multiple inheritance, so the class is restricted to extend only one class.
c)example:

```
interface Printable{

void print();

}

interface Showable{

void print();

}

class TestInterface implements Printable, Showable{

public void print(){System.out.println("Hello");}

public static void main(String args[]){

TestInterface obj = new TestInterface();
obj.print();

}

}
```

Output:
```
Hello
```

Q4]**Write a Java program to create interfaces myCafe and myTrip having methods eat() and travel() respectively. Create a class Holiday and implement multiple inheritance to override the methods eat() and travel() to print " I am eating my food" and " I am trying my route".**

```
interface myCafe{
   public void eat();
}

interface myTrip{
   public void travel();
}

class Holiday implements myTrip, myCafe{
   public void eat(){
      System.out.println("I am eating my food");
   }
   public void travel(){
      System.out.println("I am trying my route");
   }
}
public class Main
{
         public static void main(String[] args) {
            Holiday h = new Holiday();
            h.eat();
            h.travel();
         }
}
```

Output

```
I am eating my food
I am trying my route
```

Q5]Define Package and discuss access protection of class members with respect to package. In what ways can we include a package in our program? Illustrate with examples.

a)
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.

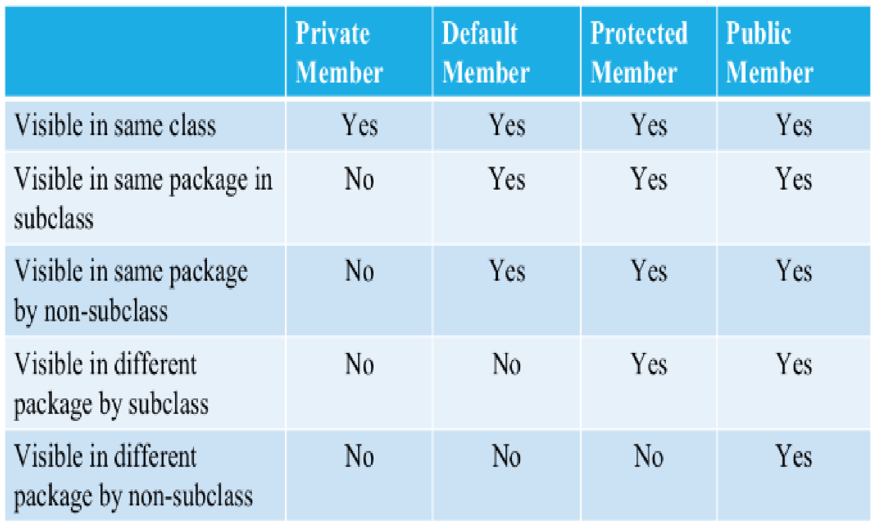
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

b) Ways to include package in java

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

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

c) Access protection of class members with respect to package

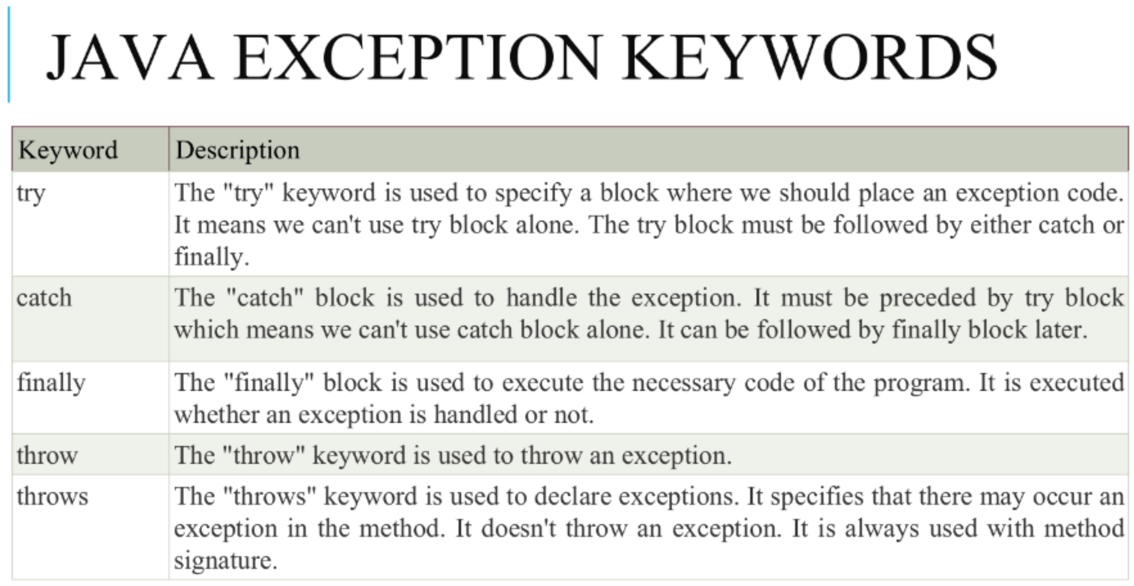


	Private Member	Default Member	Protected Member	Public Member
Visible in same class	Yes	Yes	Yes	Yes
Visible in same package in subclass	No	Yes	Yes	Yes
Visible in same package by non-subclass	No	Yes	Yes	Yes
Visible in different package by subclass	No	No	Yes	Yes
Visible in different package by non-subclass	No	No	No	Yes

Q6] **Write a Package MCA which has one class Student. Accept student detail (Name,USN,mARK) THROUGH PARAMETERIZED CONSTRUCTOR.Write display() method to display details. Create a main class which will use package and calculate total marks and percentage.**

Q7]**Explain the five keywords related to exception handling and write a program to implement all the keywords**

a) Five keywords of exception Handling



JAVA EXCEPTION KEYWORDS

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
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.
finally	The "finally" block is used to execute the necessary 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 specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Example

```
 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…
```

Q8]**Write a Java program to create a method that takes string as input and throws an exception if the string does not contain vowels**

```
public class Vowel_Check {
  public static void main(String[] args) {
    try {
      String text = "Java handling and managing exceptions ";
      // String text = "Typy gyps fly.";
      System.out.println("Original string: " + text);
      checkVowels(text);
      System.out.println("String contains vowels.");
    } catch (NoVowelsException e) {
      System.out.println("Error: " + e.getMessage());
    }
  }

  public static void checkVowels(String text) throws NoVowelsException {
    boolean containsVowels = false;
    String vowels = "aeiouAEIOU";

    for (int i = 0; i < text.length(); i++) {
      char ch = text.charAt(i);
      if (vowels.contains(String.valueOf(ch))) {
        containsVowels = true;
        break;
      }
    }
    if (!containsVowels) {
      throw new NoVowelsException("String does not contain any vowels.");
    }
  }
}
class NoVowelsException extends Exception {
  public NoVowelsException(String message) {
    super(message);
  }
}
```

Output
```
String contains vowels.
```

Output2
```
Error: String does not contain any vowels
```

Q9]What are checked and unchecked exceptions? Write a java program to illustrate single try multiple catch statements.

a) Checked and unchecked exception

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception.

1. Checked Exception

2. Unchecked Exception
3. Error

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

b)

```
public class MultipleCatchBlock1 {

public static void main(String[] args) {

try{

int a[]=new int[5];

a[5]=30/0;
}
catch(ArithmeticException e)
{
System.out.println("Arithmetic Exception occurs");
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("ArrayIndexOutOfBounds Exception occurs");
}
catch(Exception e)
{
System.out.println("Parent Exception occurs");
}
System.out.println("rest of the code");
}


}
```

Output:

```
Arithmetic Exception occurs
rest of the code
```

Q10]Write a program that handles the following exceptions using one try and multiple catch blocks

Division by Zero: Print “Invalid division”.

String parsed to a numeric variable: Print “Invalid division”.

Accessing an invalid index in string : Print “Index is invalid”

Accessing an invalid index in Array: Print “Array Index is invalid”

```
public class ProgramTen {

    public static void main(String[] args) {

        try {

            int divbyzero = 5 / 0;

            int stringIntoInt = Integer.parseInt("aniket");

            String name = "aniket";

            char wrongIndex = name.charAt(18);

            int arr[] = new int[5];

            System.out.println(arr[8]);


        } catch (ArithmeticException e) {

            System.out.println("Invalid Division");

        } catch (NumberFormatException e) {

            System.out.println("Format Mismatch");

        } catch (StringIndexOutOfBoundsException e) {

            System.out.println("Index is invalid");
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Array Index is Invalid.");
        }
    }
}
```

Output

```
Invalid Division
```