Protected Modifier and Exception Handling

1. Explain the protected access modifier in Java. In which scenarios is it most useful, and how does it differ from private and public modifiers?

Answer- The access level of a protected modifier is within the package and outside the package through child class. If we do not make the child class, it cannot be accessed from outside the package. The protected access modifier is less restrictive than private and default. Members declared as protected can be accessed within the same package or in subclasses in different packages.

2-If a class is defined in one package and a subclass is defined in another package, can the subclass access the protected members of the superclass? Provide an example to illustrate your answer.

Answer-

Yes ,subclass can access the protected member of the superclass

3-How does the protected access modifier behave in the context of inheritance and package visibility? Give a scenario where using protected would be more appropriate than using private or public.

Answer- The behaviour of protected access modifier are:

Inheritance:

- A `protected` member of a superclass can be accessed by subclasses, even if the subclass is in a different package. This allows subclasses to use or modify the protected members of their superclass.

Package Visibility:

- Within the same package, `protected` members are accessible just like package-private (default) members. This means that classes within the same package can access protected members of other classes in that package
-

```
package com.package1;
public class Parent {

    protected String protectedField = "Protected member";
    protected void protectedMethod() {
        System.out.println("This is a protected method.");
    }
}
```

```
package com.package2;

import com.package1.Parent;

public class Child extends Parent {
    public void accessProtectedMembers() {
        // Accessing protected member from the superclass
        System.out.println(protectedField);
        // Accessing protected method from the superclass
        protectedMethod();
    }
    public static void main(String[] args) {
        Child child = new Child();
        child.accessProtectedMembers();
    }
}
```

4- Describe the purpose of the try and catch blocks in Java exception handling. How does the finally block complement these, and when would you use it?

Answer-Java **try** block is used for the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Java **catch** block is used to handle the Exception by declaring the type of exception within the parameter

Java finally block is a block used to execute important code such as closing the connection, etc.