Exception handling

What is exception handling?

Exception handling is a critical aspect of modern software development, ensuring that programs can gracefully recover from unexpected situations.

1.Types of Exceptions

In any software application, exceptions can be broadly categorized into two systems.

1. Checked Exception
2. Unchecked Exception
3. Customized Exception

1.Checked Exception

Checked exceptions are exceptions that are checked at compile-time. These exceptions must be either caught or declared in the method signature using the throws keyword.

Example

- IOException
- ClassNotFoundException
- FileNotFoundException

2.Unchecked Exception

Unchecked exceptions are exceptions that are not checked at compile-time. These include runtime exceptions and errors. Unchecked exceptions typically represent programming errors, such as logic errors.

Example

- NullPointerException
- ArrayIndexOutOfBoundsException
- IllegalArgumentException

Try-Catch Blocks

In Java, try-catch blocks are used to handle exceptions gracefully. The code that might throw an exception is placed inside a try block, and the code to handle the exception is placed inside one or more catch blocks. This mechanism allows the program to continue executing even if an error occurs.

Syntax

try {

} catch (ExceptionType1 e1) {

} catch (ExceptionType2 e2) {

}

Finally Block

The finally block in Java is used to execute important code such as cleaning up resources, regardless of whether an exception is thrown or not. It is always executed after the try and catch blocks have been executed.The finally block in Java is a powerful feature for ensuring that essential cleanup code is always executed, regardless of whether an exception occurs

Syntax

try {

} catch (ExceptionType1 e1) {

} finally {

}

Declaring Exceptions with Throw Keyword

The throws keyword is used in method declarations to specify that the method may throw one or more types of exceptions. It ensures proper exception handling and propagation up the call stack, enhancing code clarity and maintainability.

Syntax

public void methodName() throws ExceptionType1, ExceptionType2 { }

Custom Exception:

```
class Shop{
int itemsNeeded;
public void setItems(int items) throws NoItemsException{
if(items==0){
throw NoItemsException("No items added");
}
this.itemsNeeded = items;
}
}

public static void main(){
Shop customerOne = new Shop();

try{
customerOne.setItems(0);
}
catch(NoItemsException exception){
System.out.println(exception);
}
}

class NoItemsException extends exception{
public NoItemsException(string message){
}
}
```