

Exception Handling

The finally Clause

Creating Custom Exception Class

The `finally` Clause

The code in a `finally` statement is always executed whether an exception occurs or not.

```
try {  
    statements;  
}  
catch (TheException ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}
```

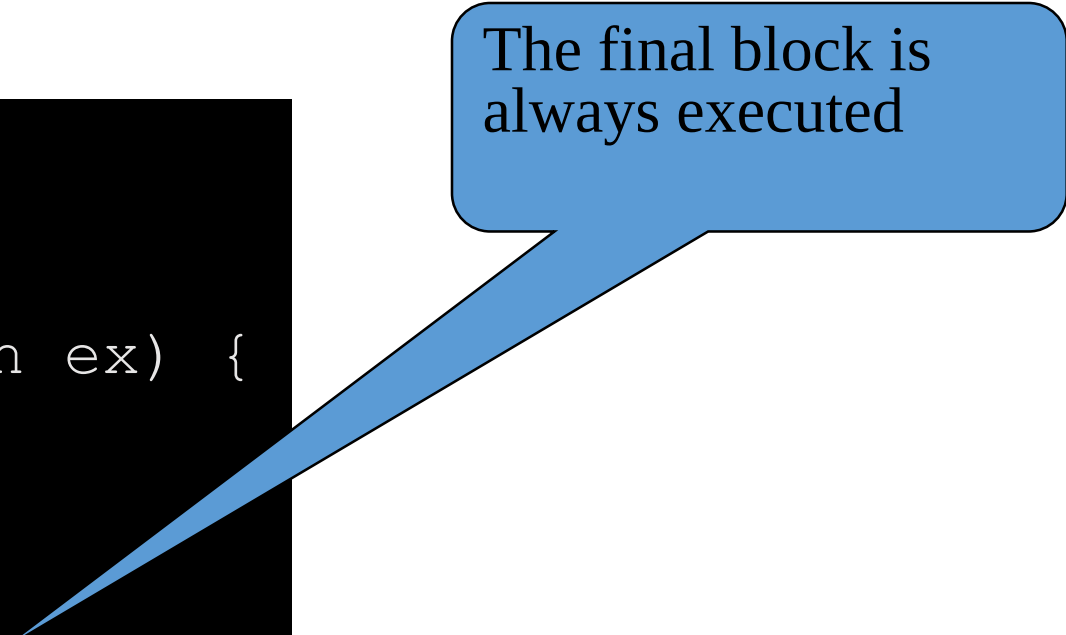
Trace a Program Execution

Suppose no
exceptions in the
statements

```
try {  
    statements;  
}  
catch (TheException ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
  
Next statement;
```

Trace a Program Execution

```
try {  
    statements;  
}  
catch (TheException ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
  
Next statement;
```



The final block is
always executed

Trace a Program Execution

```
try {  
    statements;  
}  
catch (TheException ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}
```

Next statement;

Next statement in the method is executed

Trace a Program Execution

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
  
Next statement;
```

Suppose an exception
of type Exception1 is
thrown in statement2

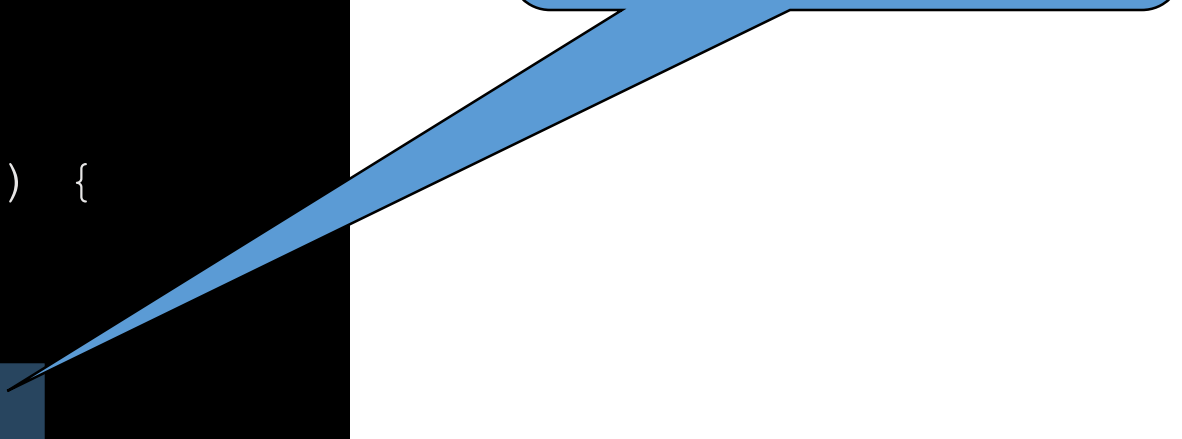
Trace a Program Execution

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
  
Next statement;
```

The exception is handled.

Trace a Program Execution

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
  
Next statement;
```



The final block is
always executed.

Trace a Program Execution

```
try {  
    statement1;  
    statement2;  
    statement3;  
}  
catch (Exception1 ex) {  
    handling ex;  
}  
finally {  
    finalStatements;  
}  
Next statement;
```

The next statement in the method is now executed.

Cautions When Using Exceptions

- Exception handling separates error-handling code from normal programming tasks, thus making programs easier to read and to modify. Be aware, however, that exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack, and propagating the errors to the calling methods.

When to Throw Exceptions

- An exception occurs in a method. If you want the exception to be processed by its caller, you should create an exception object and throw it. If you can handle the exception in the method where it occurs, there is no need to throw it.

When to Use Exceptions

When should you use the try-catch block in the code? You should use it to deal with unexpected error conditions. Do not use it to deal with simple, expected situations. For example, the following code

```
try {  
    System.out.println(refVar.toString());  
}  
catch (NullPointerException ex) {  
    System.out.println("refVar is null");  
}
```

When to Use Exceptions

is better to be replaced by

```
if (refVar != null)
    System.out.println(refVar.toString());
else
    System.out.println("refVar is null");
```

Creating Custom Exception Class

- ➡ Use the exception classes in the API whenever possible.
- ➡ Define custom exception classes if the predefined classes are not sufficient.
- ➡ Define custom exception classes by extending Exception or a subclass of Exception.

Custom Exception Class Example

```
public class InvalidRadiusException extends Exception {  
  
    public InvalidRadiusException(String message) {  
        super(message);  
        // calling the constructor of Exception  
    }  
  
}
```

Custom Exception Class Example

```
public class Circle {  
  
    private double radius;  
  
    public Circle(double r) throws InvalidRadiusException {  
        if (r < 0)  
            throw new InvalidRadiusException("The radius cannot  
                                                be negative");  
        radius = r;  
    }  
  
    public void print() {  
        System.out.println("The radius is " + radius);  
    }  
  
}
```

Custom Exception Class Example

```
import java.util.Scanner;

public class TestingInvalidRadiusException {
    public static void main(String[] args) {

        Scanner scan = new Scanner (System.in);
        double r;
        System.out.println("Enter a radius: ");
        r = scan.nextDouble();
        try
        {
            Circle c1 = new Circle(r);
            c1.print();
        }
        catch (InvalidRadiusException ex) {
            System.out.println(ex);
        }
    }
}
```