Exception-Handling Overview

Show runtime error
- Quotient
- Run

Fix it using an if statement
- QuotientWithIf
- Run

What if the runtime error occurs in a called method?
- QuotientWithException
- Run

Exception Advantages

Now you see the advantages of using exception handling. It enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

Handling InputMismatchException

By handling InputMismatchException, your program will continuously read an input until it is correct.

Exception Types

Many more classes

Many more classes

Many more classes

Many more classes

NullPointerException

IndexOutOfBoundsException

IllegalArgumentException

Many more classes

NullPointerException

IndexOutOfBoundsException

IllegalArgumentException

Many more classes

NullPointerException

IndexOutOfBoundsException

IllegalArgumentException

Many more classes
System Errors

Check the right side of the equation

Uncaught Exception
Description: Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.

Checked Exceptions vs. Unchecked Exceptions

• RuntimeException, Error and their subclasses are known as unchecked exceptions – Manifest at runtime
• All other exceptions are known as checked exceptions
  – Exceptions are checked and dealt with by the programmer
  – Exceptions are anticipated at programming time, not at runtime

Unchecked Exceptions

• In most cases, unchecked exceptions reflect programming logic errors that are not recoverable, e.g.
  – a NullPointerException is thrown if you access an object through a reference variable before an object is assigned to it
  – an IndexOutOfBoundsException is thrown if you access an element in an array outside the bounds of the array
• Unchecked exceptions can occur anywhere in the program
• These are the logic errors that should be corrected in the program
• To avoid cumbersome overuse of try-catch blocks, Java does not mandate you to write code to catch unchecked exceptions
Declaring, Throwing, and Catching Exceptions

**Declaring Exceptions**
- Every method must state the types of checked exceptions it might throw
- This is known as declaring exceptions

```java
public void myMethod() throws IOException
public void myMethod() throws IOException, OtherException
```

**Throwing Exceptions**
- When the program detects an error, the program can create an instance of an appropriate exception type (class) and throw it
- This is known as throwing an exception
- Here is an example:

```java
throw new TheException();
TheException ex = new TheException();
throw ex;
```

**Throwing Exceptions Example**

```java
/** Set a new radius * /
public void setRadius(double newRadius) throws IllegalArgumentException {
    if (newRadius >= 0)
        radius = newRadius;
    else
        throw new IllegalArgumentException("Radius cannot be negative");
}
```

**Catching Exceptions**

```java
try {
    statements; // Statements that may throw exceptions
} catch (Exception1 exVar1) {
    handler for exception1;
} catch (Exception2 exVar2) {
    handler for exception2;
} ... catch (ExceptionN exVar3) {
    handler for exceptionN;
}
```
Catch or Declare Checked Exceptions

Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must invoke it in a try-catch block or declare to throw the exception in the calling method. For example, suppose that method p1 invokes method p2 and p2 may throw a checked exception (e.g., IOException), you have to write the code as shown in (a) or (b).

```java
void p1() {
    try {
        p2();
    }
    catch (IOException ex) {
        ...
    }
}
```

Example: Declaring, Throwing, and Catching Exceptions

- Objective: This example demonstrates declaring, throwing, and catching exceptions by modifying the setRadius method in the Circle class defined in Chapter 8. The new setRadius method throws an exception if radius is negative.

```java
TestCircleWithException CircleWithException
```