

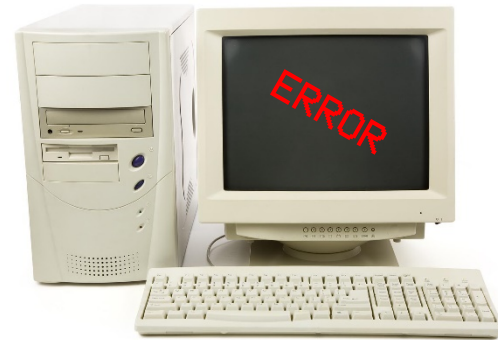
File Edit View Help

Exceptions

Part 01

Exceptions

- Exceptions signals an *exceptional* occurrence during run-time
- Handles run-time errors by allowing the program to *crash gracefully* and keep executing
- Exceptions are Objects
 - These Objects have an “exception message”
- “Throwing” an exception is when an exception object is created
- “Handling” an exception is when special code detects and deals with the exceptional occurrence



Exceptions

- 3 Major Parts to Exceptions
 1. Creating Exceptions
 2. Throwing (Using) Exceptions
 3. Handling Exceptions



Creating Exceptions

- In Java there are several predefined exceptions
 - Exception (most general)
 - NullPointerException
 - IndexOutOfBoundsException
 - IOException
- Creating a specific kind of Exception involves inheriting from one of the predefined Exceptions
- Only write the Constructors
 - Make sure to use “super” to construct the superclass Exception
 - Set the exception message
 - Do not override “getMessage”

Syntax for Creating an Exception

```
public class <<id>> extends <<an Exception>>
{
    <<constructors>>;
}
```

Example

```
public DivideByZeroException extends Exception
{
    public DivideByZeroException()
    {
        super("Attempted to Divide by Zero");
    }
    public DivideByZeroException(String msg)
    {
        super(msg);
    }
}
```

Throwing Exceptions

- When a method could cause an Exception, then then programmers need to be notified to handle it
- The reserved word “throws” is used in the method signature to indicate the method could cause an exception
- Each exception is listed by their identifier and are separated using a comma

Syntax for a Method that throws an Exception

```
<<scope>> <<return type>> <<method id>> (<<parameters>>)  
throws <<List of Exceptions>>  
{  
    <<method body>>  
}
```

Example

```
public double evaluate(char op, double n1, double n2)  
throws DivideByZeroException, UnknownOpException  
{  
    ...  
}
```

Throwing Exceptions

- In a method that *throws* exceptions there should be cases where that kind of exception happens
- The reserved word “throw” is used when an exception occurs
 - Method signature uses “throws”
 - Method body uses “throw”
- Follow “throw” by then constructing an instance of that kind of exception

Syntax for Throwing the Exception

```
throw new <<Exception Constructor>>
```

Example

```
...  
//Inside of method evaluate  
...  
throw new DivideByZeroException();  
...
```

Handling Exceptions

- Methods that *throws* exceptions must be handled in a “try-catch” block
- The method that could cause the exception must be within the body of the try-block
 - Otherwise the method would cause a syntax error
- The exception that is handled must be declared in the arguments of the catch-block
 - Exception type followed by an identifier
- The exception is then handled in the body of the catch-block
 - Usually a good idea to print the exception message using either “getMessage” or “printStackTrace”

Handling an Exception

```
try
{
    <<Method that throws the Exception>>
}
catch(<<Exception type>> <<id>>)
{
    <<Handle the Exception>>
}
```

Example

```
try
{
    ...
    result = evaluate(nextOp, result, nextNumber);
    ...
}
catch(DivideByZeroException e)
{
    e.printStackTrace();
}
```


Handling Exceptions

- If a method causes an exception in the try-block then the program immediately *jumps* to the corresponding catch-block
- After the exception has been handled the program continues after the try-catch block
- A try-catch block can only have 1 try-block and may have 1 or more catch-blocks
- Multiple Catch-blocks must be ordered from most specific exception to least specific exception
 - Otherwise causes an unreachable code syntax error
 - Most general exception is “Exception”
- With multiple catch-blocks the most appropriate catch-block runs corresponding to the exception that was thrown

Syntax for Handling a Multiple Exception

```
try
{
    <<Method that throws the Exceptions>>
}
catch(<<Most Specific Exception type>> <<id>>)
{
    <<Handle the Most Specific Exception>>
}
...
catch(<<Most General Exception type>> <<id>>)
{
    <<Handle the Most General Exception>>
}
```

Handling Exceptions

- If a method causes an exception in the try-block then the program immediately *jumps* to the corresponding catch-block
- After the exception has been handled the program continues after the try-catch block
- A try-catch block can only have 1 try-block and may have 1 or more catch-blocks
- Multiple Catch-blocks must be ordered from most specific exception to least specific exception
 - Otherwise causes an unreachable code syntax error
 - Most general exception is “Exception”
- With multiple catch-blocks the most appropriate catch-block runs corresponding to the exception that was thrown

Example

```
try
{
    ...
    result = evaluate(nextOp, result, nextNumber);
    ...
}
catch(DivideByZeroException e)
{
    e.printStackTrace();
}
catch(UnknownOpException e)
{
    e.printStackTrace();
}
catch(Exception e)
{
    e.printStackTrace();
}
```

Handling Exceptions

- A “finally” block can be optionally added after a sequence of catch-blocks
- The code in the finally-block will execute whether or not an exception is thrown

Finally Block Syntax

```
try
{
    <<Method that throws the Exception>>
}
catch(<<Exception type>> <<id>>)
{
    <<Handle the Exception>>
}
finally
{
    <<code that will execute with or without exceptions>>
}
```

Example

```
try
{
    ...
    result = evaluate(nextOp, result, nextNumber);
    ...
}
//Catches
finally
{
    System.out.println("result = " + result);
}
```

Example

Simple Calculator

- Problem: We must create a simple calculator program
- Keeps track of a resulting value
- Performs the operations
 - Addition
 - Subtraction
 - Multiplication
 - Division
- User provides input via the console
- Input follows <<operator>> <<value>>
 - Example “+ 3”
- Must handle a variety of exceptions while keeping the program running