CISC 3120

C16b: Exception Handling

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Exceptions
- Catch and handle exceptions (try/catch)
- Forward exceptions
  - Specify requirement for methods (throws)
- Customize exceptions and types of exceptions
- Checked and unchecked exception
- Some Best Practice

What may can go wrong?

- A simple Java expression evaluator

```
public class SimpleExpr {
   public static void main(String[] args) {
     int d1 = Integer.parseInt(args[0]);  int d2 = Integer.parseInt(args[2]);  int result;
     switch(args[1]) {
     case "+":
        result = d1 + d2;  System.out.println(String.join(" ",  args) + " = " + result);  break;
     case "-":
        result = d1 - d2;  System.out.println(String.join(" ",  args) + " = " + result);  break;
     case "*":
        result = d1 * d2;  System.out.println(String.join(" ",  args) + " = " + result);  break;
     case "/":
        result = d1 / d2;  System.out.println(String.join(" ",  args) + " = " + result);  break;
     }
   }
}
```

Things can go wrong

```
$ java SimpleExpr ↵
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
$ java SimpleExpr  1.0 + 2.0 ↵
Exception in thread "main" java.lang.NumberFormatException: For input string: "1.0"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
$ java SimpleExpr  3 / 0 ↵
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:12)
```

Exceptions

- Java uses *exceptions* to handle errors and other exceptional events.
- Exception: exceptional situations at runtime
    - An event that occurs during the execution of a program that disrupts the normal flow of instructions.
    - Examples
        - Division-by-zero
        - Access an array element that does not exist
- When the exception situation occurs, Java throws an "exception".

Throws an Exception

- The method where an error occurs creates an “Exception” object and hands it off to the Java runtime
- Exception object, contains
  - Information about error
    - Its type and the state of the program when the error occurred..

Call Stack

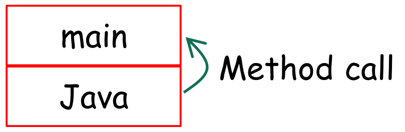
- Java runtime attempts to find objects and methods to handle it
  - An ordered list of methods that had been called to get to the method where the error occurred.
- The list of methods is known as the call stack

Example Exception

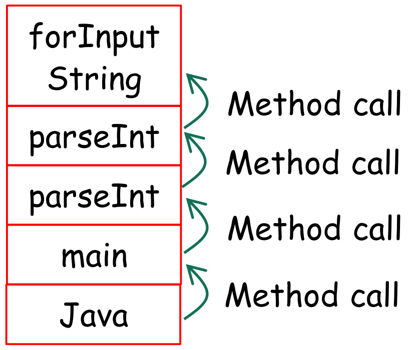
```
$ java SimpleExpr ↵
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
$ java SimpleExpr  1.0 + 2.0 ↵
Exception in thread "main" java.lang.NumberFormatException: For input string: "1.0"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
$ java SimpleExpr  3 / 0 ↵
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:12)
```

Example Call Stack

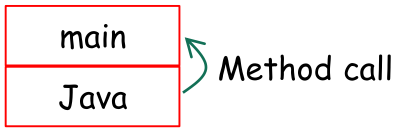
```
$ java SimpleExpr ↵
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
            at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
```



```
$ java SimpleExpr  1.0 + 2.0 ↵
Exception in thread "main" java.lang.NumberFormatException: For input string: "1.0"
            at java.lang.NumberFormatException.forInputString(Unknown Source)
            at java.lang.Integer.parseInt(Unknown Source)
            at java.lang.Integer.parseInt(Unknown Source)
            at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:3)
```

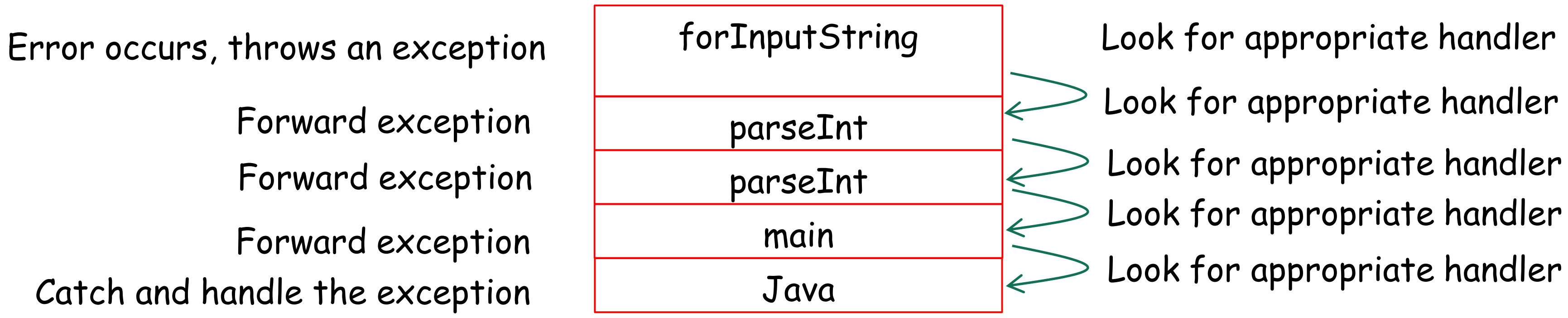


```
$ java SimpleExpr  3 / 0 ↵
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at edu.cuny.brooklyn.cisc3120.simpleexpr.SimpleExpr.main(SimpleExpr.java:12)
```



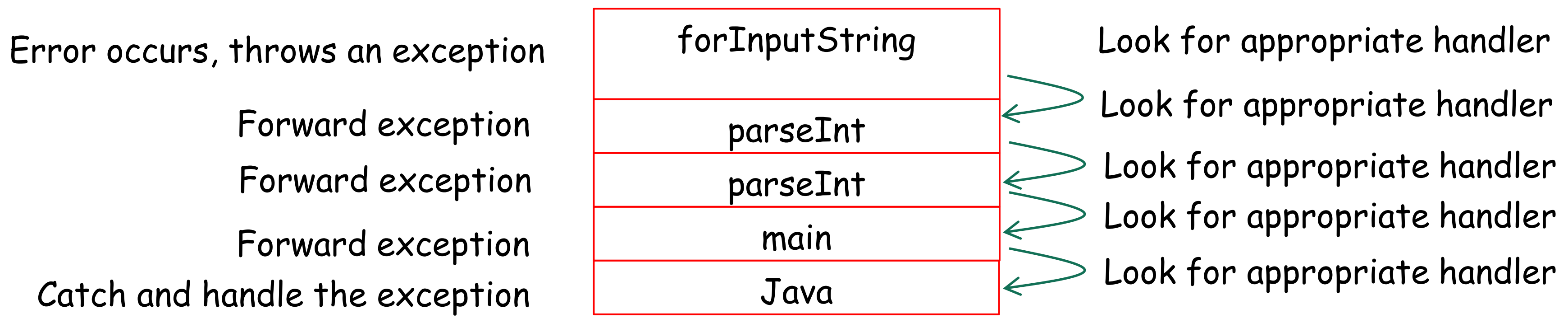
Example Exception Throws and Catch

- When an exception happens, what does JVM (Java runtime) do?


Error occurs, throws an exception
Forward exception
Forward exception
Forward exception
Catch and handle the exception
forInputString
parseInt
parseInt
main
Java
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler

Example Exception Throws and Catch

- When an exception happens, what does JVM (Java runtime) do?
- Application developers may, sometimes must decide whether to forward or to catch/handle the exceptions


Error occurs, throws an exception
Forward exception
Forward exception
Forward exception
Catch and handle the exception
forInputString
parseInt
parseInt
main
Java
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler

Questions?

- Exception?
- Call stack?

Catch or Forward?

- Java code must either to catch or forward exceptions
  - A try statement that catches the exception.
    - The try must provide a handler for the exception.
  - A method can be specified to forward an exception
    - The method must provide a throws clause that lists the exception.

Catching and Handling Exceptions

- Use the try-catch blocks
- Use the try-catch-finally blocks
  - To be discussed when we discuss File I/O
- Use the try with resource blocks
  - To be discussed with we discuss File I/O

Try-catch Example: One Exception

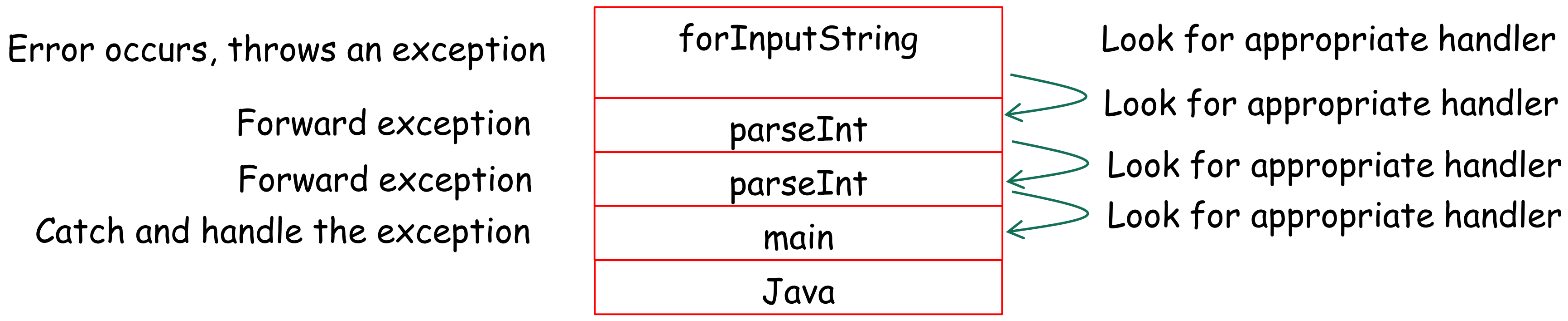
- Try and catch a single exception

```
try {
    d1 = Integer.parseInt(args[0]);
} catch (NumberFormatException e) {
    System.out.println("SimpleExprImproved must take two integer operants.“ + args[0] + " is not an integer.");
    System.exit(-1);
}
```

```
try {
    result = d1 / d2;
    System.out.println(String.join(" ",  args) + " = " + result);
} catch (ArithmeticException e) {
    System.out.println("The divisor cannot be zero.");
    System.exit(-1);
}
```

Try-catch Example: One Exception: Call Stack

```
try {
    d1 = Integer.parseInt(args[0]);
} catch (NumberFormatException e) {
    System.out.println("SimpleExprImproved must take two integer operants.“ + args[0] + " is not an integer.");
    System.exit(-1);
}
```


Error occurs, throws an exception
Forward exception
Forward exception
Catch and handle the exception
forInputString
parseInt
parseInt
main
Java
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler
Look for appropriate handler

Try-Catch Example: More Exception

- Try and catch more than one exceptions

```
try {
    d1 = Integer.parseInt(args[0]);
} catch (NumberFormatException e) {
    System.out.println("SimpleExprImproved must take two integer operants.“ + args[0] + " is not an integer.");
    System.exit(-1);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Usage: SimpleExprImproved <integer> <+|-|*|/> <integer>"
    return;
}
```

```
try {
    d1 = Integer.parseInt(args[0]);
} catch (NumberFormatException | ArrayIndexOutOfBoundsException  e) {
    System.out.println(“Either “ + args[0] + " is not an integer.“
         + “ or use it correctly: Usage: SimpleExprImproved <integer> <+|-|*|/> <integer>");
    System.exit(-1);
}
```

Questions?

- Catch and handle an exception?

Forward Exception

- What if we do want to let a method further up the call stack handle the exception?
- Specifying the Exceptions thrown by a method

Forward Exception: Example

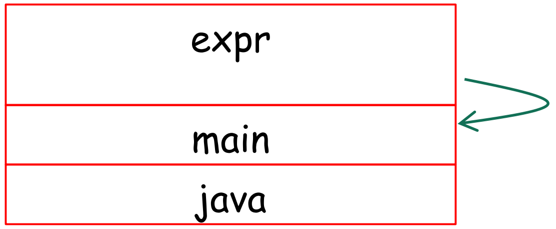
```
public class SimpleExprThrows {
    public static void main(String[] args) {
        ......
        try {
            result = expr(d1, d2, args[1]); System.out.println(d1 + args[1] + d2 + "=" + result);
        } catch(ArithmeticException e) {
            System.out.println("The divisor is 0.");
        }
    }
    public static int expr(int d1, int d2, String operator) throws ArithmeticException {
        int result = 0;
        switch(operator) {
        case "+":  result = d1 + d2; break;
        case "-":  result = d1 - d2; break;
        case "*":  result = d1 * d2; break;
        case "/":  result = d1 / d2;
        }
        return result;
    }
}
```

Forward Exception: Example: Call Stack

```
public class SimpleExprThrows {
    public static void main(String[] args) {
        ......
        try { result = expr(d1, d2, args[1]); ….} catch(ArithmeticException e) { … }
    }
    public static int expr(int d1, int d2, String operator) throws ArithmeticException {
        int result = 0;
        switch(operator) {
         ......
        case "/":  result = d1 / d2;
        }
        return result;
    }
}
```

Error occurs, throws an exception

Catch and handle the exception



Look for appropriate handler

Look for appropriate handler

Questions?

- Forward exceptions via method specification.

Is there a difference?

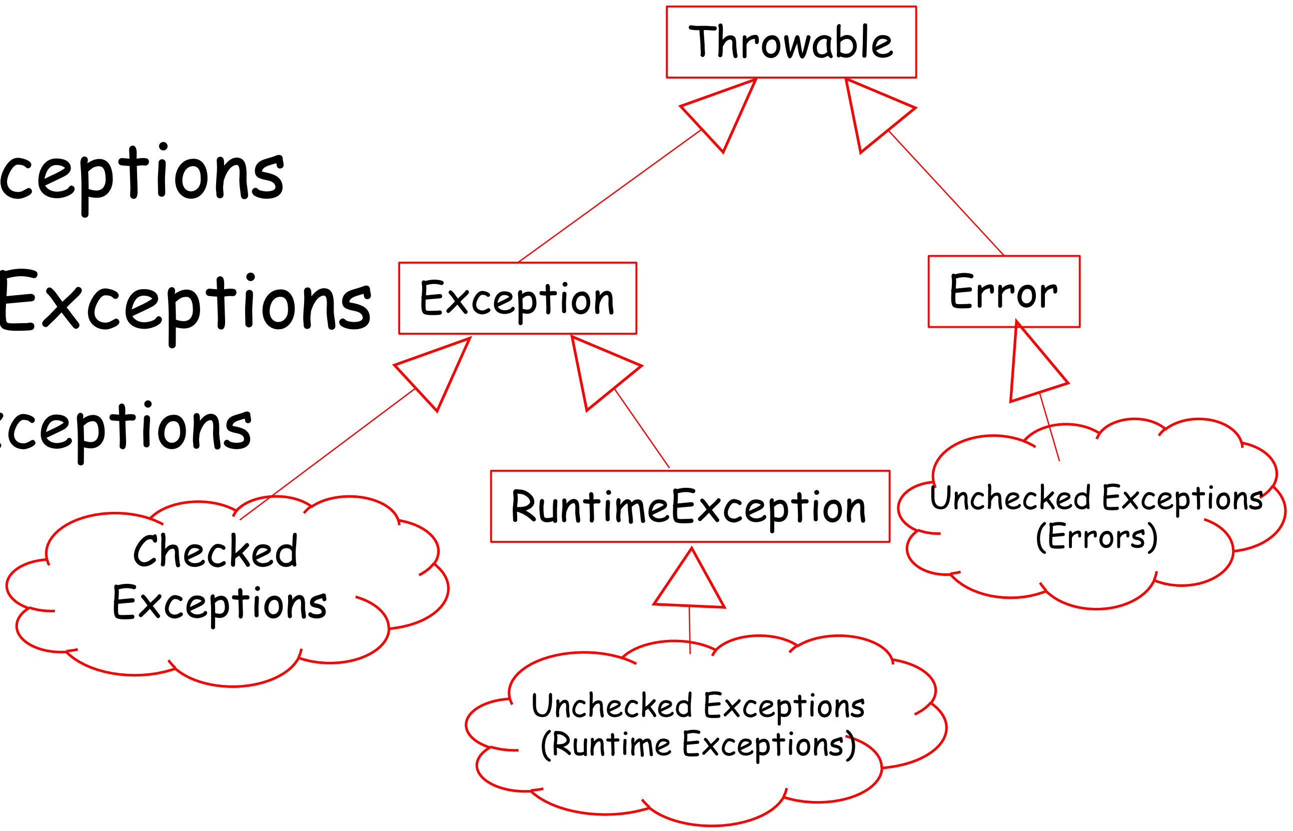
- Forward exception: wait a minute, what's really different?
    - Let's examine the call stacks of the two

```
public class SimpleExprThrows {
   ……
   public static int expr(int d1, int d2, String
operator) {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2; break;
      case "-":  result = d1 - d2; break;
      case "*":  result = d1 * d2; break;
      case "/":  result = d1 / d2;
      }
      return result;
   }
}
```

```
public class SimpleExprThrows {
   ……
   public static int expr(int d1, int d2, String operator)
throws ArithmeticException {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2; break;
      case "-":  result = d1 - d2; break;
      case "*":  result = d1 * d2; break;
      case "/":  result = d1 / d2;
      }
      return result;
   }
}
```

Checked and Unchecked Exceptions, and Errors

- Java has three types of exceptions and errors
- Checked Exceptions
- Unchecked Exceptions
  - RuntimeExceptions
  - Errors


Throwable
Exception
Error
RuntimeException
Checked Exceptions
Unchecked Exceptions (Runtime Exceptions)
Unchecked Exceptions (Errors)

Checked Exceptions

- Checked Exceptions
  - The class Exception and any subclasses that are not also subclasses of RuntimeException are checked exceptions.
- Semantics
  - Exceptional conditions that a well-written application should anticipate and recover from.
- Subject to the Catch or Specify Requirement.
  - You must either forward it explicitly (via specifying requirement when define a method) or handle it.

Unchecked Exceptions

- RuntimeException and Error and their subclasses
- Not subject to the Catch or Specify Requirement
  - Forwarded automatically, if not caught and handled or forwarded explicitly (via specif
- Semantics
  - The application usually cannot anticipate or recover from.
  - Errors
    - The Error class or subclass of Error
  - RuntimeException
    - The RuntimeException class or subclass of RuntimeException

Error and RuntimeException

- Errors
    - Semantics: exceptional conditions that are external to the application.
    - Example:
        - hardware failure: disk I/O error after a file is open, and read will fail and cannot recover from it.
            - Your file is on a USB drive. While your app is reading the file, you unplug the drive.
- Runtime exceptions
    - Semantics: exceptional conditions that are internal to the application.
    - Example:
        - logical error: passed a string in a non-integer form to Integer.parseInt() that cannot recover from it (but you may avoid it, if not the first time)

Is there a difference?

- ArithmeticException is a RuntimeException, it will be forwarded automatically if not explicitly

```
public class SimpleExprThrows {
   ……
   public static int expr(int d1, int d2, String
operator) {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2; break;
      case "-":  result = d1 - d2; break;
      case "*":  result = d1 * d2; break;
      case "/":  result = d1 / d2;
      }
      return result;
   }
}
```

```
public class SimpleExprThrows {
   ……
   public static int expr(int d1, int d2, String operator)
throws ArithmeticException {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2; break;
      case "-":  result = d1 - d2; break;
      case "*":  result = d1 * d2; break;
      case "/":  result = d1 / d2;
      }
      return result;
   }
}
```

Questions

- Checked exceptions, runtime exceptions, and errors

Customize or Raise an Exception?

- Any code can throw an exception.
- Use the throw statement.
- Purposes
    - Raises an exception
    - Customize an exception
    - Create new types of exceptions

Can Anything Else Go Wrong?

- Can anything go wrong in addition to ArithmeticException?

```
public class SimpleExprThrows {
    ……
    public static int expr(int d1, int d2, String operator) throws ArithmeticException {
        int result = 0;
        switch(operator) {
        case "+":  result = d1 + d2; break;
        case "-":  result = d1 - d2; break;
        case "*":  result = d1 * d2; break;
        case "/":  result = d1 / d2;
        }
        return result;
    }
}
```

Can Anything Else Go Wrong?

- How about we do this? What’s wrong?

```
$ java SimpleExpr 2 ^ 3 ↲
2^3=0
```

```
public class SimpleExprThrows {
    ……
    public static int expr(int d1, int d2, String operator) throws ArithmeticException {
        int result = 0;
        switch(operator) {
        case "+":  result = d1 + d2; break;
        case "-":  result = d1 - d2; break;
        case "*":  result = d1 * d2; break;
        case "/":  result = d1 / d2;
        }
        return result;
    }
}
```

Raise an Exception: Example

```
public class SimpleExprThrows {
   public static void main(String[] args) { ...
      try {
         result = expr(d1, d2, args[1]);  System.out.println(d1 + args[1] + d2 + "=" + result);
      } catch(ArithmeticException e) {  System.out.println("The divisor is 0.");
      } catch(IllegalArgumentException e) { System.out.println(e.getMessage()); }
   }

   public static int expr(int d1, int d2, String operator) throws ArithmeticException, IllegalArgumentException {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2;  break;
      case "-":  result = d1 - d2;  break;
      case "*":  result = d1 * d2; break;
      case "/":  result = d1 / d2;  break;
      default:
         throw new IllegalArgumentException("Operator " + operator + " is not supported.");
      }
      return result;
   }
}
```

Commonly Reused Exceptions

- Use of standard exceptions are generally preferred (Bloch, J., 2008)

Exception	Occasion for Use
IllegalArgumentException	Non-null parameter value is inappropriate
IllegalStateException	Object state is inappropriate for method invocation
NullPointerException	Parameter value is null where prohibited
IndexOutOfBoundsException	Index parameter value is out of range
ConcurrentModificationException	Concurrent modification of an object has been detected where it is prohibited
UnsupportedOperationException	Object does not support method

Customize an Exception: Example

```
public class SimpleExprThrows {
   public static void main(String[] args) { ...
      try {
         result = expr(d1, d2, args[1]);  System.out.println(d1 + args[1] + d2 + "=" + result);
      } catch(ArithmeticException e) {  System.out.println("The divisor is 0.");
      } catch(IllegalArgumentException e) { System.out.println(e.getMessage()); }
   }

   public static int expr(int d1, int d2, String operator) throws ArithmeticException, IllegalArgumentException {
      int result = 0;
      switch(operator) {
      case "+":  result = d1 + d2;  break;
      case "-":  result = d1 - d2;  break;
      case "*":  result = d1 * d2; break;
      case "/":  try { result = d1 / d2; } catch (ArithmetricException e) {
                              throw new ArithmeticException("Attempt to evaluate " + d1 + "/" + d2);
               }  break;
      default:
         throw new IllegalArgumentException("Operator " + operator + " is not supported.");
      }
      return result;   }   }
```

Subtype an Exception

- You may extends an Exception class
- Example

```
public class SimpleExprException extends RuntimeException {
		SimpleExprException(String msg) {
				super(msg);
		}
}
```

Questions

- Raise exception
- Customer exception
- Create new types of exception

Some Best Practices

- Do throw specific Exceptions

```
throw new RunTimeException("Exception at runtime");
```

✗

- Throw early, catch late.
  - better to throw a checked exception than to handle the exception poorly.
- Use exception only for exception situations

```
if (args.length != 3) {
   System.out.println("Usage …");
}
```

✓

```
try {
  d1 = Integer.parseInt(args[2]);
} catch (ArrayIndexOutOfBoundsException e) {
  System.out.println("Usage …");
}
```

✗

Questions

- Murphy's law
  - Anything that can go wrong will go wrong.

- Exceptions
- Catch exceptions (try/catch)
- Specify requirement for methods (throws)
- Declare and throw exceptions (throw)
- Checked and unchecked exception
- Some Best Practice

Assignment

- Practice Assignment