CIS 351-Data Structure-Exception
Feb 11, 2020

Dr. Farzana Rahman

Syracuse University

What is wrong with following code

```
1  /**************************************************
2  * Return the mean, or -1 if the array has length 0.
3  **************************************************/
4  public static double mean(double[] numbers)
5        {
6           double sum = 0;
7           double result;
8
9           if (numbers == null || numbers.length == 0)
10          {
11             result = -1;
12          }
13          else
14          {
15             for (int i = 0; i < numbers.length; i++)
16             {
17                sum += numbers[i];
18             }
19             result = sum / numbers.length;
20          }
21          return result;
22       }
```

- Sometimes there is no appropriate return value that can be used to indicate an error has occurred. (Let's use exceptions to improve this code...)
- Exceptions provide flexibility in deciding where a particular problem should be handled. If an exception occurs, a method may:
  - "Pass the buck" by using the throws keyword. The exception will be handled somewhere higher-up in the call stack.
  - Deal with the exception using a try/catch block.

```java
    public static double mean(double[] numbers)
    {
        if (numbers == null || numbers.length == 0)
        {
            throw new IllegalArgumentException("Invalid array.");
        }

        double sum = 0;

        for (int i = 0; i < numbers.length; i++)
        {
            sum += numbers[i];
        }

        return sum / numbers.length;
    }
```

Motivation

- We seek robust programs
- When something unexpected occurs
  - Ensure program detects the problem
  - Then program must do something about it
- Need to check for problem where it could occur
- When condition does occur
  - Have control passed to code to handle the problem

Overview

- Exception
  - Indication of problem during execution
- Uses of exception handling
  - Process exceptions from program components
  - Handle exceptions in a uniform manner in large projects
- A method detects an error and throws an exception
  - Exception handler processes the error
  - Uncaught exceptions yield adverse effects
    - Might terminate program execution

What Exactly Is an Exception?

- An exception means that an action member cannot complete the task it was supposed to perform as indicated by its name.

Checked Exceptions

- Nonruntime (*checked exceptions*)*:*
  - These exceptions occur outside of the runtime system
  - Input / Output exceptions
  - Caught or specified by the system compiler

Runtime Exceptions

- Runtime (*unchecked exceptions*)*:*
  - Arithmetic Exceptions: dividing by zero
  - Null Pointer Exceptions: attempting to reference a method of a null pointer
  - Class Cast Exception: casting incompatible object, super or subclass types
  - Index Out of Bounds Exception: accessing a index value outside of an array range
  - Typically result from logical errors

Java built-in exceptions

- ArithmeticException
- ArrayIndexOutOfBounds
- ArrayStore
- ClassCast
- IllegalArgument
- IllegalState
- IllegalThreadState

Exception Handling Terms

- throw – to generate an exception or to describe an instance of an exception
- try – used to enclose a segment of code that may produce a exception
- catch – placed directly after the try block to handle one or more exception types
- finally – optional statement used after a try-catch block to run a segment of code regardless if a exception is generated

Handling Exceptions

```
try {
   <code segment that may
      throw an exception..>
} catch (ExceptionType) {
   <exception handler..>
} catch (ExceptionType) {
   <exception handler..>
} finally {
   <optional code segment..>
}
```

Multiple catch statements

- Once a try statement has been used to enclose a code segment, the catch can be used to handle one or more specific exception types.

- By defining catch statements for specific exception types, a more accurate handling of the exception can be tailored to the programs needs

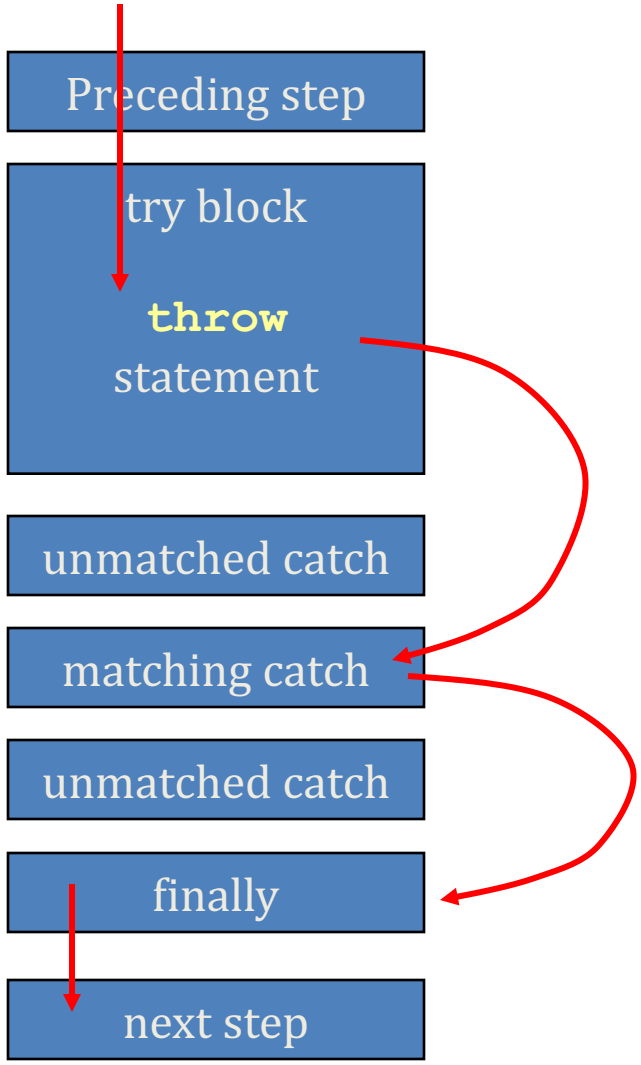
Multiple catch statements

```
try {
   <code segment that may
      throw an exception..>
} catch (IOException e) {
   System.out.println(e.getMessage());
} catch (FileNotFoundException e) {
   System.out.println("File Not Found!");
}
```

`finally` Block

- The purpose of the optional finally statement will allow the execution of a segment of code regardless if the try statement throws an exception or executes successfully
- Executes whether or not an exception is thrown in the corresponding `try` block or any of its corresponding `catch` blocks
- Will not execute if the application exits early from a `try` block via method `System.exit`
- Typically contains resource-release code

Sequence of Events for `finally` clause



Throw vs Throws

- The `throw` keyword is similar to `return`
  - Ends the method
  - Sends a result (exception object) to the caller
- The `throws` keyword is similar to a method's return type specification.

```
1     public double someMethod(double arg) throws SomeCheckedException
2     {
3         if (arg == 0.0)
4         {
5             throw new SomeCheckedException("Bad argument.");
6         }
7
8         return 10.0;
9     }
```

Checked vs Unchecked Exception

- The `throws` keyword is only required for "checked exceptions"
  - They must be handled within the method with a try-catch block OR
  - Declared thrown with `throws` (passing the buck)
- "Unchecked exceptions" inherit from `RuntimeException`
  - Often result from programming errors, not exceptional circumstances.

Example

```
1           fileName = "NONEXISTENTFILE.txt";
2           System.out.print("A ");
3           try
4           {
5               System.out.print("B ");
6               file = new File(fileName);
7               scanner = new Scanner(file);
8               System.out.print("C ");
9
10          }
11          catch (FileNotFoundException e)
12          {
13              System.out.print("D ");
14          }
15          finally
16          {
17              System.out.print("E ");
18          }
19          System.out.print("F ");
```

Example

- What exception will be thrown when the following application is executed? Why?

```
private static double[] sales;

public static void main(String[] args)

{
    double[] percentages;
    percentages = toPercentages(sales);
}
```

A `NullPointerException` will be thown because sales was not initialized.

Java Exception Hierarchy

- All exceptions inherit either directly or indirectly from class `Exception`

- Exception classes form an inheritance hierarchy that can be extended

Class Circle

```
public class Circle
{
  public double x, y;     // centre of the circle
  public double r;        // radius of circle

  public Circle(double radius)
  { this. r = radius; }

   //set and get methods are present, but removed for
   space constraint

  public double circumference()
  {
      return 2*3.14*r;
  }
  public double area()
  {
      return 3.14 * r * r;
  }
}
```

Comparing objects (1)-Radius

```
Circle c1 = new Circle (5.0);
Circle c2 = new Circle (5.0);
System.out.print(c1.equals(c2)); // we want True
```

```
                                        c2
public boolean equals(Circle other)
{           c1                          c2
     if (this.getRadius() == other.getRadius())
          return true;  5.0                    5.0
     else
          return false;
}
```

Comparing objects (2)-Area

```
Circle c1 = new Circle (5.0);
Circle c2 = new Circle (5.0);
System.out.print(c1.equals(c2)); // we want True
```

c2

```
public Circle equals(Circle other)
{
```

c1 78.5 c2 78.5

```
     if (this.getArea() == other.getArea())
          return true;
     else
          return false;
}
```

Comparing objects (3)-Area

```
Circle c1 = new Circle (5.0);
Circle c2 = new Circle (10.0);
System.out.print(c1.equals(c2)); // we want false
```

c2

```
public Circle equals(Circle other)
{
```

c1 5.0 c2 10.0

```
     if (this.getRadius() == other.getRadius())
          return true;
     else
          return false;
}
```

Understanding Main

```
Public Test {
    public static void main (String[] args)
    {
        if (args.length > 0)
        {
                System.out.print (args[0]);
                System.out.print (args[1]);
                System.out.print (args[2]);
        }
    }
}
```

From Terminal: ***java*** **Test I Love Programming**