SENG1110/SENG6110 Object Oriented Programming

Lecture 10

Exception

Outline

- Exceptions in Java
- Handling Exceptions in Java
- `try/catch` block
- Predefined Exception Classes
- Declaring Exceptions (Passing the Buck)
- Kinds of Exceptions
- Multiple Throws and Catches
- The `finally` Block
- Rethrowing an Exception
- Case Study: A Line-Oriented Calculator

Exceptions

- An occurrence of an undesirable situation that can be detected during program execution
- For example
  - division by zero
  - trying to open an input file that does not exist
  - an array index that goes out of bounds

Handling Exceptions within a Program

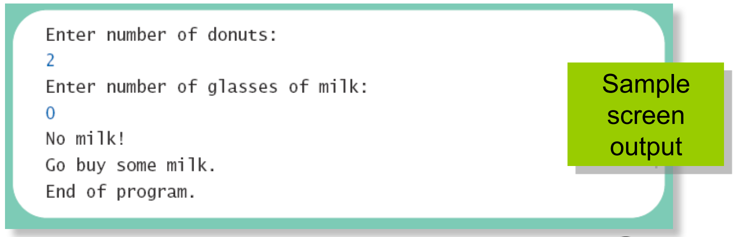
- Can use an `if` statement to handle an exception.
- However, suppose that division by zero occurs in more than one place within the same block.
  - In this case, using if statements may not be the most effective way to handle the exception.

Handling Exceptions in Java

- When an exception occurs, an object of a particular exception `class` is created.
- Java provides a number of exception classes to effectively handle certain common exceptions, such as:
  - Division by zero
  - Invalid input
  - File not found

Exceptions in Java

- Consider a program to assure us of a sufficient supply of milk
- View CodeSamplesWeek10_exception
  `class GotMilk`



```
Enter number of donuts:
2
Enter number of glasses of milk:
0
No milk!
Go buy some milk.
End of program.
```

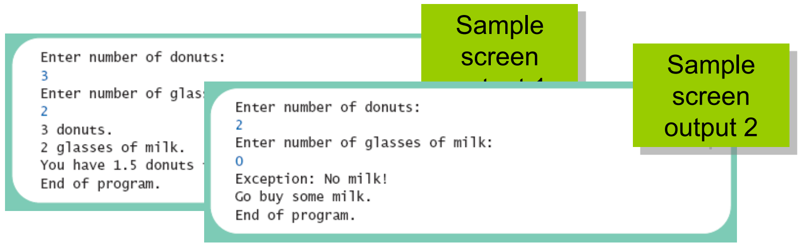
Sample screen output

Handling Exceptions in Java

- While such occurrences are errors, they should be predicted by the programmer and provision made for them, so that the program is able to handle the exception and does not crash

Exceptions in Java

- Now we revise the program to use exception-handling
- View CodeSamplesWeek10_exception
  `class ExceptionDemo1`



```
Enter number of donuts:
3
Enter number of glas
2
3 donuts.
2 glasses of milk.
You have 1.5 donuts
End of program.
```

Sample screen

```
Enter number of donuts:
2
Enter number of glasses of milk:
0
Exception: No milk!
Go buy some milk.
End of program.
```

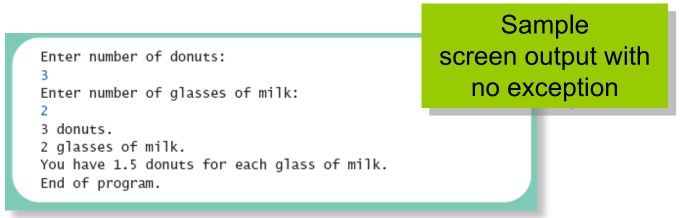
Sample screen output 2

Exceptions in Java

- Note try `block`
  - Contains code where something could possibly go wrong
  - If it does go wrong, we *throw an exception*
- Note `catch` block
  - When exception thrown, `catch` block begins execution
  - Similar to method with parameter
  - Parameter is the thrown object

Exceptions in Java

- Note flow of control when no exception is thrown
- View CodeSamplesWeek10_exception
  `class ExceptionDemo2`

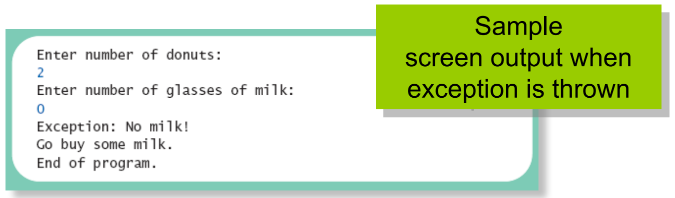


Sample screen output with no exception

```
Enter number of donuts:
3
Enter number of glasses of milk:
2
3 donuts.
2 glasses of milk.
You have 1.5 donuts for each glass of milk.
End of program.
```

Exceptions in Java

- Note flow of control when exception IS thrown
- View CodeSamplesWeek10_exception
  `class ExceptionDemo3`



Sample screen output when exception is thrown

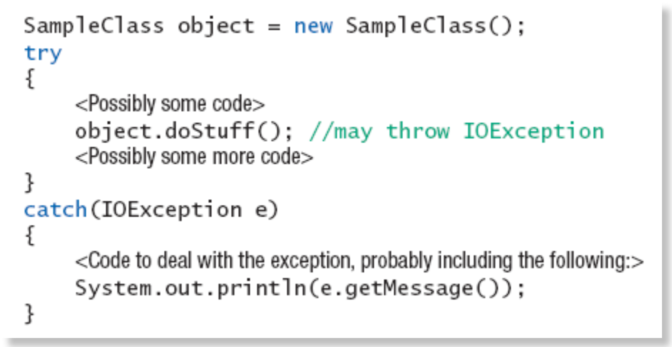
```
Enter number of donuts:
2
Enter number of glasses of milk:
0
Exception: No milk!
Go buy some milk.
End of program.
```

Predefined Exception Classes

- Java has predefined exception classes within Java Class Library
  - Can place method invocation in `try` block
  - Follow with `catch` block for this type of exception
- Example classes
  - `BadStringOperationException`
  - `ClassNotFoundException`
  - `IOException`
  - `NoSuchMethodException`

Predefined Exception Classes

- Example code



```
SampleClass object = new SampleClass();
try
{
    <Possibly some code>
    object.doStuff(); //may throw IOException
    <Possibly some more code>
}
catch(IOException e)
{
    <Code to deal with the exception, probably including the following:>
    System.out.println(e.getMessage());
}
```

Example

- For example, an array index of inappropriate size could be handled:

a.length = 3
positions are 0, 1 and 2

```
int[] a = {10, 20, 30};

try
{
System.out.println(a[a.length]);
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("Error: index out of bounds");
}
```

Example

- For example, an input integer of incorrect format could be handled:

```
int number;
String str = keyboard.readLine();

try
{
        number = Integer.parseInt(str);
}
catch (NumberFormatException e)
{
        number = 0;
        System.out.println ("Error: bad format for number:"+ str);
}
```

Example - Multiple Exceptions

```
int[] a = {10, 0, 30};

try
{
System.out.println(a[0]/a[1]);
}
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println ("Error: index out of bounds");
}
catch (ArithmeticException e)
{
     System.out.println ("Error: attempt to divide by 0");
}
```

Example - A Generic Exception

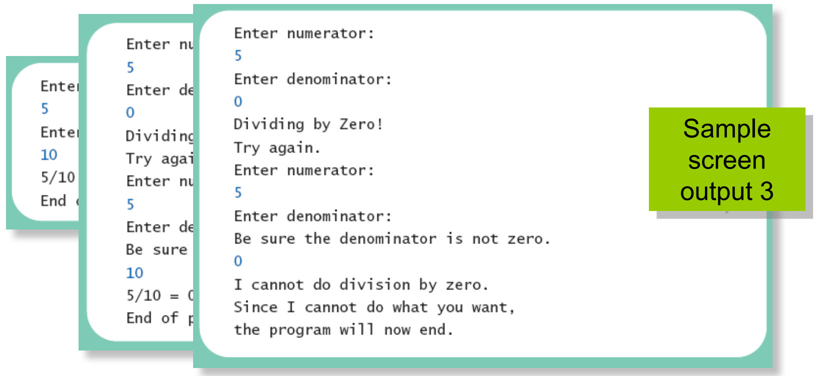
```
int[] a = {10, 0, 30};

try
{
        System.out.println (a[0] / a[1]);
}
catch (Exception e)
{
 System.out.println ("Error:\n" + e.toString());
}
```

- Specific exceptions are extends from the generic Exception class
  - a divide by zero exception and
  - an array out of bounds exception are
  - still instances of the Exception class as well as being instances of their own specific classes.

Defining Your Own Exception Classes

- Different runs of the program



```
Ente
5
Ente
10
5/10
End
```

```
Enter nu
5
Enter de
0
Dividing
Try agai
Enter nu
5
Enter de
Be sure
10
5/10 = 0
End of p
```

```
Enter numerator:
5
Enter denominator:
0
Dividing by Zero!
Try again.
Enter numerator:
5
Enter denominator:
Be sure the denominator is not zero.
0
I cannot do division by zero.
Since I cannot do what you want,
the program will now end.
```

Defining Your Own Exception Classes

- Must be derived class of some predefined exception class
  - Text uses classes derived from class Exception
- View CodeSamplesWeek10_exception
  `class DivideByZeroException extends Exception`
- View CodeSamplesWeek10_exception
  `class DivideByZeroDemo`

Defining Your Own Exception Classes

- Note method `getMessage` defined in exception classes
  - Returns string passed as argument to constructor
  - If no actual parameter used, default message returned
- The type of an object is the name of the exception class

Defining Your Own Exception Classes

Guidelines

- Use the `Exception` as the base class
- Define at least two constructors
  - Default, no parameter
  - With `String` parameter
- Start constructor definition with call to constructor of base class, using `super`
- Do not override inherited `getMessage`

Declaring Exceptions

- Note syntax for throws clause

```
public Type Method_Name(Parameter_List) throws List_Of_Exceptions
Body_Of_Method
```

- Note distinction
  - Keyword `throw` used to throw exception
  - Keyword `throws` used in method heading to declare an exception

Declaring Exceptions

- Consider method where code throws exception
  - May want to handle immediately
  - May want to delay until something else is done
- Method that does not catch an exception
  - Notify programmers with `throws` clause
  - Programmer then given responsibility to handle exception

Declaring Exceptions

- If a method throws exception and exception not caught inside the method
  - Method ends immediately after exception thrown
- A throws clause in overriding method
  - Can declare fewer exceptions than declared
  - But not more
- View CodeSamplesWeek10_exception `class DoDivision`

Kinds of Exceptions

- In most cases, exception is caught …
  - In a **catch** block … or
  - Be declared in **throws** clause
- But Java has exceptions you do not need to account for
- Categories of exceptions
  - Checked exceptions
  - Unchecked exceptions

Kinds of Exceptions

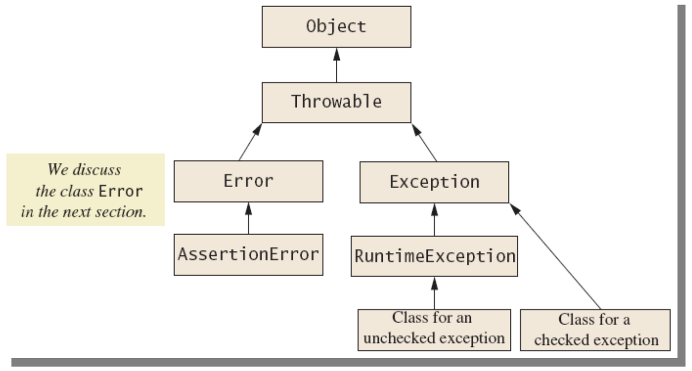
- Examples why unchecked exceptions to are thrown
  - Attempt to use array index out of bounds
  - Division by zero
- Uncaught runtime exception terminates program execution

Kinds of Exceptions

- *Checked* exception
  - represent invalid conditions in areas outside the immediate control of the program
  - Must be caught in **catch** block
  - Or declared in **throws** clause
- *Unchecked* exception
  - Also called *run-time* (defects in the program - bugs)
  - Need not be caught in **catch** block or declared in **throws**
  - Exceptions that coding problems exist, should be fixed

Kinds of Exceptions

- Figure 9.1 Hierarchy of the predefined exception classes


Object
Throwable
We discuss the class Error in the next section.
Error
Exception
AssertionError
RuntimeException
Class for an unchecked exception
Class for a checked exception

Multiple Throws and Catches

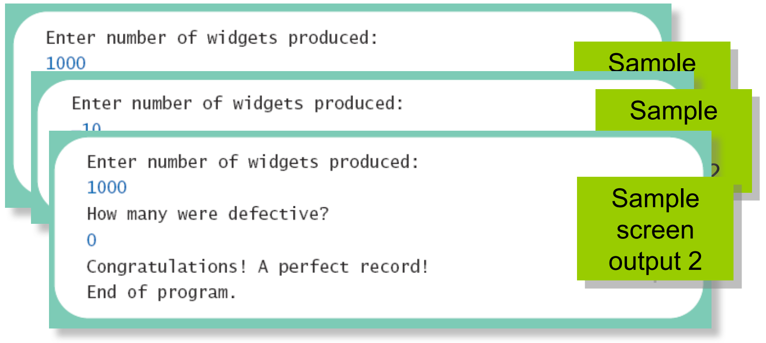
- A try block can throw any number of exceptions of different types
- Each catch block can catch exceptions of only one type
  - Order of catch blocks matter
- View CodeSamplesWeek10_exception `class TwoCatchesDemo`
- View CodeSamplesWeek10_exception `class NegativeNumberException`

The `finally` Block

- Possible to add a `finally` block after sequence of `catch` blocks
- Code in `finally` block executed
  - Whether or not execution thrown
  - Whether or not required `catch` exists

Multiple Throws and Catches

- Note multiple sample runs



```
Enter number of widgets produced:
1000
```

```
Enter number of widgets produced:
-10
```

```
Enter number of widgets produced:
1000
How many were defective?
0
Congratulations! A perfect record!
End of program.
```

The class `Exception` and the Operator `instanceof`

```
try
{
    System.out.print("Line 4: Enter dividend: ");
    System.out.flush();
    dividend =
            Integer.parseInt(keyboard.readLine());
    System.out.println();

    System.out.print("Line 8: Enter divisor: ");
    System.out.flush();
    divisor
            = Integer.parseInt(keyboard.readLine());
    System.out.println();

    quotient = dividend / divisor;
    System.out.println("Line 13: quotient = "
                               + quotient);
}
catch(Exception eRef)
{
    if(eRef instanceof ArithmeticException)
        System.out.println("Line 16: Exception "
                               + eRef.toString());
    else if(eRef instanceof NumberFormatException)
            System.out.println("Line 18: Exception "
                               + eRef.toString());
    else if(eRef instanceof IOException)
            System.out.println("Line 20: Exception "
                               + eRef.toString());
}
```

Exception-Handling Techniques

- Terminate program
  - Output appropriate error message upon termination
- Fix error and continue
  - Repeatedly get user input
  - Output appropriate error message until valid value is entered
- Log error and continue
  - Write error messages to file and continue with program execution

Case Study – to be checked later

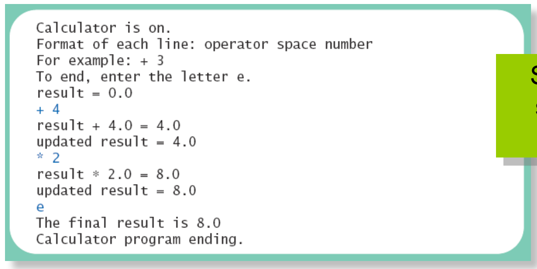
- Proposed initial methods
  - Method to `reset` value of `result` to zero
  - Method to `evaluate` result of one operation
  - Method `doCalculation` to perform series of operations
  - Accessor method `getResult`: returns value of instance variable `result`
  - Mutator method `setResults`: sets value of instance variable `result`

Case Study – to be checked later

- A Line-Oriented Calculator
  - Should do addition, subtraction, division, multiplication
  - Will use line input/output
- User will enter
  - Operation, space, number
  - Calculator displays result

Case Study – to be checked later

- View CodeSamplesWeek10_exception `class UnknownOpException`
- View first CodeSamplesWeek10_exception `class PreLimCalculator`



```
Calculator is on.
Format of each line: operator space number
For example: + 3
To end, enter the letter e.
result = 0.0
+ 4
result + 4.0 = 4.0
updated result = 4.0
* 2
result * 2.0 = 8.0
updated result = 8.0
e
The final result is 8.0
Calculator program ending.
```

Sample screen output

Case Study – to be checked later

- Final version adds exception handling
- Ways to handle unknown operator
  - Catch exception in method `evaluate`
  - Let `evaluate` throw exception, catch exception in `doCalculation`
  - Let `evaluate`, `doCalculation` both throw exception, catch in `main`
- Latter option chosen

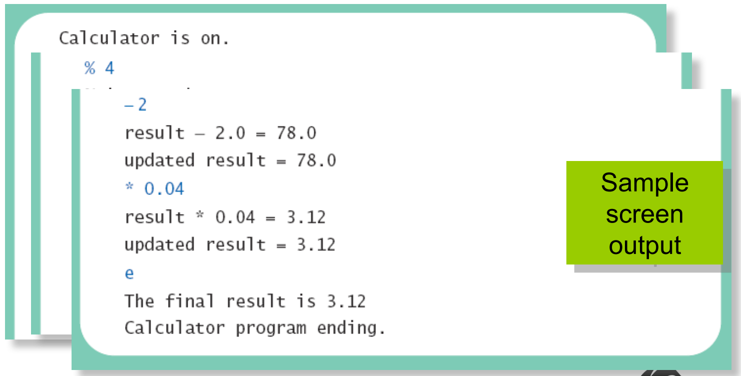
Your task

- Read
  - Lecture slides
  - Chapter 9
- Exercises
  - MyProgrammingLab
  - Computer lab exercises

☺ Have fun!!!

Case Study – to be checked later

- View CodeSamplesWeek10_exception
  `class Calculator`



```
Calculator is on.
  % 4
    -2
    result - 2.0 = 78.0
    updated result = 78.0
    * 0.04
    result * 0.04 = 3.12
    updated result = 3.12
    e
    The final result is 3.12
    Calculator program ending.
```

Sample screen output