Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120A - Fall 2004

Lecture Unit 13

More Control Structures

Exceptions

- Unusual or erroneous situations are represented in Java by objects known as *exceptions*
- An exception is "thrown" by a program and can be "caught" and appropriately handled if desired
- An error is like an exception but usually represents an unrecoverable situation
- Options for your program:
  - Not handle an exception at all
  - Handle the exception where it occurs
  - Handle the exception at another point in the program

Exception Messages

```
public class Zero {

   public Zero() {
      int num = 10;
      int denom = 0;

      System.out.println(num / denom);
      System.out.println("This text will not be printed.");
   }

   public static void main(String[] args) {
      Zero z = new Zero();
   }

}
```

what type of exception and why it occurred

```
>java Zero
java.lang.ArithmeticException: / by zero
        at Zero.<init>(Zero.java:7)
        at Zero.main(Zero.java:12)
```

method call trace

The try Statement

```
try {
  … // some statements
} catch (ExceptionTypeA errA) {
  …  // handle errA
} catch (ExceptionTypeB errB) {
  … // handle errB
}
```

- Can have zero or more catch blocks
- errA and errB are exception objects
- If particular exception type is not "caught" it is automatically propagated

Try-Catch Example

```
public static void main(String[] args) {
   BufferedReader in = new BufferedReader(
                new InputStreamReader(System.in));

   String line = "";
   boolean ok = false;
   while (!ok) {
      try {
         System.out.print("Please enter part number: ");
         line = in.readLine();
         // line should be something like A78930

         char ch = line.charAt(0);
         String substr = line.substring(1, 6);
         int val = Integer.parseInt(substr);

         ok = true;
      } catch (IOException e) {
         System.out.println(e);
         System.out.println("You messed up somehow!");
      } catch (StringIndexOutOfBoundsException e) {
         System.out.println(e);
         System.out.println("Not enough characters?");
      } catch (NumberFormatException e) {
         System.out.println(e);
         System.out.println("Didn't type a number?");
      }
   }
}
```


Optional finally Clause

- Defines a section of code that is executed no matter how the try block is exited
  - If no exception: statements executed after "try" block is complete
  - If exception: appropriate "catch" block is executed, then the statements in the "finally" block

```
try {
  … // some statements
} catch (ExceptionTypeA errA) {
  …  // handle errA
} finally {
  … // executed no matter what
}
```

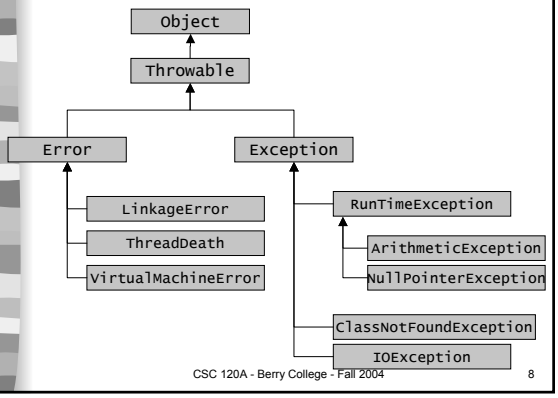
Generating ("Raising") Exceptions

- Use the throw statement:
  throw <object-expression>;
- Most often, you define your own exception classes, since the predefined ones are thrown automatically by the Java runtime

```
public class InvalidPartNumberException extends Exception {
   public InvalidPartNumberException() {
      super();
   }

   public InvalidPartNumberException(String msg) {
      super(msg);
   }
}
```


Exception Class Hierarchy (incomplete)



Types of Exceptions

- Unchecked – derived from RunTimeException
- Checked – must be caught by a method or else must be listed in a throws clause of the method (e.g. IOException)
- (Errors are liked unchecked exceptions
  - Usually, should not be caught and do not require listing in a throws clause

Switch Control Statement

- Control structure convenient for multiway branches
- Similar to nested if statements
- Value of a switch expression (which must be an integral expression) determines which branch executes

```
switch (<expression>) {
  case <constant-expression>:
          … // statements
          break;    // optional
  default:
          … // statements
          break; // optional
}
```


Switch Example

```
public static void printResultA(char res) {
   if (res == 'A' || res == 'B') {
      System.out.println("Good work");
   }
   else if (res == 'C') {
      System.out.println("Average Work");
   }
   else if (res == 'D' || res == 'F') {
      System.out.println("Poor Work");
      numInTrouble++;
   }
   else {
      System.out.println("Invalid grade");
   }
}
```

```
public static void printResultB(char res) {
   switch (res) {
      case 'A':
      case 'B':
         System.out.println("Good Work");
         break;
      case 'C':
         System.out.println("Average Work");
         break;
      case 'D':
      case 'F':
         System.out.println("Poor Work");
         numInTrouble++;
         break;
      default:
         System.out.println("Invalid grade");
         break;
   }
}
```


Another Example

```
public static void printResultC(int grade) {
   int category = grade / 10;

   String str = "";
   switch (category) {
      case 10: str += "very ";
      case 9:  str += "good work"; break;
      case 8:  str += "above average"; break;
      case 7:  str += "average"; break;
      case 6:  str += "below average"; break;
      default: str += "not passing";
   }
   System.out.println(str);
}
```

Loops

- Java provides three types of loops:
  - while

```
while (<boolean-expression>)
  // statement
```

  - do-while

```
do
  // statement
while (<boolean-expression>);
```

  - For

```
for (<init> ; <boolean-expression> ; <update> )
  // statement
```

- Watch out for semi-colons
- Remember, statement can be a block { … }