# COT 3530: Data Structures


Giri Narasimhan

ECS 389; Phone: x3748

giri@cs.fiu.edu


www.cs.fiu.edu/~giri/teach/3530Spring04.html

# Evaluation

- Midterm & Final Exams
- Programming Assignments
- Class Participation

# What is Java?

- Java programming language
  - object-oriented
  - similar to C++
- Java platform
  - Java Virtual Machine (JVM)
  - Application Programming Interface (API)
- Created and licensed by Sun Microsystems
- Both compiled and interpreted
  - compiler produces Java Bytecodes
  - bytecodes are executed by Java Virtual Machine
  - programs run slightly slower than native code (such as that produced by C++)

# What is Java?

- Java Development Kit (JDK)
  - compiler, runtime system, debugger, documentation
- Java Virtual Machine (JVM)
  - executes Java bytecode programs
- Java API
  - a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*.

# How Can Java be Used?

- Write software on one platform and run it on another.
  - applications
- Create programs to run within a web browser.
  - applets
- Develop server-side applications for online forums, stores, polls, processing HTML forms, and more.
- Write applications for cell phones, two-way pagers, and other consumer devices.

# Programming Environment

- JDK 1.3.1
- JCreator is our editor and Integrated Development Environment (IDE)
- Filenames
  - case-sensitive
  - Source files: .java
  - Compiled code: .class
- Project files
  - All java source files must be in same directory

# JCreator Files

- Workspace file
  - Hello.jcw
- Project file
  - Hello.jcp
- Source Code
  - Hello.java
- List of source files
  - src_hello.txt
- Compiled code (class file)
  - Hello.class

- A workspace may contain multiple projects
- A project may contain multiple java source files

# Simple Java Hello - 1

```
// Hello.java
public class Hello {
  public static void main(String[] args)
  {
    System.out.println("Hello, COP 3337");
  }
}
```

# Simple Java Hello - 2

- Create the project
  - File / New / Empty project
  - Project name: Hello
  - (Workspace created automatically)
- Create the source file
  - File / New / Java file
  - File name: Hello
- Build the project
  - Build / Compile project (F7)
- Execute the project
  - Build / Execute project (F5)

# Figure 1.2
The eight primitive types in Java

| PRIMITIVE TYPE | WHAT IT STORES | RANGE |
|---|---|---|
| byte | 8-bit integer | −128 to 127 |
| short | 16-bit integer | −32,768 to 32,767 |
| int | 32-bit integer | −2,147,483,648 to 2,147,483,647 |
| long | 64-bit integer | $-2^{63}$ to $2^{63} - 1$ |
| float | 32-bit floating-point | 6 significant digits ( $10^{-46}$, $10^{38}$ ) |
| double | 64-bit floating-point | 15 significant digits ( $10^{-324}$, $10^{308}$ ) |
| char | Unicode character | |
| boolean | Boolean variable | false and true |

 

# Primitive Types

- 8 primitive types: byte, short, int, long, float, double, char, boolean
- Non-primitive types are all reference types
  - reference types are simply pointers
  - assignments to reference types
  - declaring an object ≠ creating an object
  - garbage collection
  - reference types as parameters of methods
  - casting
- String type: concatenation, length, comparison, sub-string, conversion

# Static Declarations

- Storage allocation for static objects, fields, methods.
- 2 ways to invoke static fields & methods: using an object or using the class name.

# Arrays

- Reference types; explicitly created using new statement.
- Index starts at 0.
- Arrays have length field.
- Array assignment ≠ array copy.
- Array copy done using clone()
- multi-dimensional arrays
- Dynamic arrays – automatic using ArrayList

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class ReadStrings
{
   public static void main( String [ ] args )
   {
      String [ ] array = getStrings( );
      for( int i = 0; i < array.length; i++ )
         System.out.println( array[ i ] );
   }

   // Read an unlimited number of String; return a String [ ]
   public static String [ ] getStrings( )
   {
      BufferedReader in = new BufferedReader( new InputStreamReader( System.in ) );
      String [ ] array = new String[ 5 ];
      int itemsRead = 0;
      String oneLine;

      System.out.println( "Enter any number of strings, one per line; " );
      System.out.println( "Terminate with empty line: " );

      try
      {
         while( ( oneLine = in.readLine( ) ) != null && !oneLine.equals( "" ) )
         {
            if( itemsRead == array.length )
               array = resize( array, array.length * 2 );
            array[ itemsRead++ ] = oneLine;
         }
      }
      catch( IOException e )
      {
         System.out.println( "Unexpected IO Exception has shortened amount read" );
      }

      System.out.println( "Done reading" );
      return resize( array, itemsRead );
   }
```

Figure 2.6, 2.7, page 42-43

Implementing arrays of Strings

Figure 2.7, page 43

```
// Resize a String[ ] array; return new array
    public static String [ ] resize( String [ ] array, int newSize )
    {
        String [ ] original = array;
        int numToCopy = Math.min( original.length, newSize );

        array = new String[ newSize ];
        for( int i = 0; i < numToCopy; i++ )
            array[ i ] = original[ i ];
        return array;
    }
}
```

Same program, but using class ArrayList

Figure 2.8, page 44

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.ArrayList;
public class ReadStringsWithArrayList
{
   public static void main( String [ ] args )
   {
      ArrayList array = getStrings( );
      for( int i = 0; i < array.size( ); i++ )
         System.out.println( array.get( i ) );
   }

   // Read an unlimited number of String; return an ArrayList
   public static ArrayList getStrings( )
   {
      BufferedReader in = new BufferedReader( new InputStreamReader( System.in ) );
      ArrayList array = new ArrayList( );
      String oneLine;

      System.out.println( "Enter any number of strings, one per line; " );
      System.out.println( "Terminate with empty line: " );

      try
      {
         while( ( oneLine = in.readLine( ) ) != null && !oneLine.equals( "" ) )
            array.add( oneLine );
      }
      catch( IOException e )
      {
         System.out.println( "Unexpected IO Exception has shortened amount read" );
      }

      System.out.println( "Done reading" );
      return array;
   }
}
```

# Exceptions & Errors

- An exception is an object that is thrown from the site of an error and can be caught by an appropriate exception handler.
- Separating the handler from error detection makes the code easier to read and write. Do not use exception as a "cheap" goto statement. Better to pass it on to calling procedure.
- More reliable error recovery without simply exiting.
- User-defined exceptions can be created or thrown.
- The try region is a guarded region from which errors can be caught by exceptions.

- Errors are virtual machine problems. OutOfMemoryError, InternalError, UnknownError are examples of errors.
- Errors are uncrecoverable and should not be caught.

# Figure 2.12
Common standard run-time exceptions

| Standard Run-time Exception | Meaning |
|---|---|
| ArithmeticException | Overflow or integer division by zero. |
| NumberFormatException | Illegal conversion of String to numeric type. |
| IndexOutOfBoundsException | Illegal index into an array or String. |
| NegativeArraySizeException | Attempt to create a negative-length array. |
| NullPointerException | Illegal attempt to use a null reference. |
| SecurityException | Run-time security violation. |

# Figure 2.13
Common standard checked exceptions

| Standard Checked Exception | Meaning |
|---|---|
| java.io.EOFException | End-of-file before completion of input. |
| java.io.FileNotFoundException | File not found to open. |
| java.io.IOException | Includes most I/O exceptions. |
| InterruptedException | Thrown by the Thread.sleep method. |

# Input/Output

- Streams are used for I/O
- Terminal I/O treated in the same way as File I/O.
- Predefined streams System.in, System.out, System.err
- readLine and StringTokenizer are useful methods for formatted input; they are part of java.io.BufferedReader and java.util.StringTokenizer respectively.