CMSC 132:
Object-Oriented Programming II

Java Constructs

Department of Computer Science

University of Maryland, College Park

Overview

- Stream I/O
- Initialization Blocks
- Variable Initialization
- Annotations
- BitSet class
- Two-dimensional array

Stream Input/Output

- Stream
  - A connection carrying a sequence of data (ordered sequence of bytes)
- Streams can be associated
  - Files
  - Memory
  - Other Strings
- Java provides many classes for handling streams
  - Some handle data that consists of characters (e.g., text files)
  - Some handle data that consists of raw bytes (e.g., binary files)
  - Some buffer information
- By combining different classes we can define a stream with the characteristics we want

Using Streams

- Opening a stream
  - Connects program to external data
  - Location of stream specified at opening
  - Only need to refer to stream
- Usage
  1. import java.io.* ;
  2. Open stream connection
  3. Use stream → read and / or write
     - Catch exceptions if needed
  4. Close stream
- Examples
  - See fileExamples package

Initialization Block

- **Definition**
  - Block of code used to initialize static & instance variables for class
- **Motivation**
  - Enable complex initializations for static variables
    - Control flow
    - Exceptions
  - Share code between multiple constructors for same class

Initialization Block Types

- **Static initialization block**
  - Code executed when class loaded
- **Initialization block**
  - Code executed when each object created (at beginning of call to constructor)
- **Example**

```
class foo {
    static { A = 1; }       // static initialization block
    { A = 2; }              // initialization block
}
```

Variable Initialization

- **Variables may be initialized**
  - At time of declaration
  - In initialization block
  - In constructor
- **Order of initialization**
  1. Declaration, initialization block
     (in the same order as in the class definition)
  2. Constructor

Variable Initialization – Example

```
class Foo {
    static { A = 1; }       // static initialization block
    static int A = 2;       // static variable declaration
    static { A = 3; }       // static initialization block
    { B = 4; }              // initialization block
    private int B = 5;      // instance variable declaration
    { B = 6; }              // initialization block
    Foo() {                 // constructor
       A = 7;
       B = 8;
    }                       // now A = 7, B = 8
}           // initializations executed in order of number
```

Annotations

- Annotation – Java construct that allow us to add validity constraints to Java Classes
- Validity constraint example
  - A instance variable cannot assume a negative value
  - A parameter can not be null
  - A method in a class must override a method in its superclass
- Syntax
  - at-sign (@) followed by annotation type and a parenthesized list of element-value pairs
- Example

  @DefaultAnnotationForParameters(NonNull.class)
- You can ignore annotations in the code distribution for class projects

Reviewing Bit-Operations

- and

x	11010
y	10110
x and y	10010

- or

x	11010
y	10110
x and y	11110

- xor

x	11010
y	10110
x and y	01100

- Java Bitwise operators
  - & and
  - | or
  - ^ exclusive or
  - ~ complement

BitSet Class

- Implements a vector of bits where the bits of the set are indexed by nonnegative integers
- Methods
  - BitSet() – New bit set
  - BitSet(int nbits) – Bit set large enough to represent bits with indices from 0 through nbits – 1
  - and(BitSet set) – Performs logical and between the current object and the set parameter (current object is updated with the result)
  - or(BitSet set) – Performs logical or between the current object and the set parameter (current object is updated with the result)
  - cardinality() – Returns number of bits set to 1
  - flip(int bitIndex) – Sets the bit at the specified index
  - get(int bitIndex) – Returns true if the bit at bitIndex is set; false otherwise
  - length() – Index of the highest set bit + 1. It returns zero if the BitSet contains no bits set.
  - size() – Number of bits space used by the BitSet to represent bit values
  - toString() – For every bit set, the decimal representation of that index is included in the result.
- Example (See Computers.java)

Two-Dimensional Arrays of Primitives

- Each row in a two-dimensional array is an array
- The rows can have different lengths
- Defining a primitive array where rows have the same length

```
int [ ][ ] data = new  int[3][4];
```

- Defining a primitive data array where rows have different lengths (ragged array)

```
int [ ][ ] ragged = new  int[2][];
ragged[0] = new int[3];
ragged[1] = new int[1];
```

Two-Dimensional Arrays of Objects

- Each row in a two-dimensional array is an array
- The rows can have different lengths
- Defining an array where rows have the same length

```
String [ ][ ] data  = new  String[3][4];
```

- Important: Keep in mind we have created a two-dimensional array of references to String objects. No String object is present yet.
- We can also have ragged arrays
- Example (See Roster.java)