

CMSC 132: Object-Oriented Programming II



**Nelson Padua-Perez
William Pugh**

**Department of Computer Science
University of Maryland, College Park**

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)**

Project #1

- **CVS repository information**

- **Host: linuxlab.csic.umd.edu**

- **Repository Path:**

- [/afs/csic.umd.edu/users/pugh/cs132/cs132XXX](http://afs.csic.umd.edu/users/pugh/cs132/cs132XXX)**

where **cs132XXX** is your linuxlab account which you can find through **grades.cs.umd.edu**

- **Demo of playing the game**

- **Brief overview of what students are expected to implement**

- **Ask students to go over the open policy available on the class web site**

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)

Modern Software Development

- **Why do we want to study the software development process?**
- **To understand**
 - **Software development problems**
 - **Why software projects fail**
 - **Impact of software failures**
 - **How to develop better software**
- **Software Engineering (from Wikipedia)**

Field that creates and maintains software applications by applying technologies and practices from computer science, project management, engineering, application domains, and other fields.

Software Development Problems

- **Software is expensive**
 - **Cost per line of code increasing**
(while hardware costs drop)
- **Software is late (schedule overruns)**
- **Software cost more (cost overruns)**
- **Software is difficult to use**
- **Software is difficult to understand**
- **Software is missing features**
- **Software is too slow**

Software Projects Fail

- Anywhere from 25-50% of custom software fail
- Latest example
 - Jan 13, 2005, LA Times
 - “A new FBI computer program designed to help agents share information to ward off terrorist attacks may have to be scrapped, forcing a further delay in a four-year, half-billion-dollar overhaul of its antiquated computer system... Sources said about \$100 million would be essentially lost if the FBI were to scrap the software...”

Software Contributing to Real Failures

- **1990 AT&T long distance calls fail for 9 hours**
 - **Wrong location for C break statement**
- **1996 Ariane rocket explodes on launch**
 - **Overflow converting 64-bit float to 16-bit integer**
- **1999 Mars Climate Orbiter crashes on Mars**
 - **Missing conversion of English units to metric units**



Impact of Software Failures Increasing

- **Software becoming part of basic infrastructure**
 - **Software in cars, appliances**
 - **Business transactions moving online**
- **Computers becoming increasingly connected**
 - **Failures can propagate through internet**
 - **Internet worms**
 - **Failures can be exploited by others**
 - **Viruses**
 - **Spyware**

Why Is Software So Difficult?

- **Complexity**
 - **Software becoming much larger**
 - Millions of line of code
 - Hundreds of developers
 - **Many more interacting pieces**
- **Length of use**
 - **Software stays in use longer**
 - Features & requirements change
 - Data sets increase
 - Can outlast its creators

Software Size

- **Small**
 - 1-2 programmers, < 3000 lines of code
- **Medium**
 - 2-5 programmers, < 20,000 lines of code
- **Large**
 - 5-20 programmers, < 100,000 lines of code
- **Very large**
 - 20-200 programmers, < 1 million lines of code
- **Extremely large**
 - > 200 programmers, > 1 million lines of code

SLOC

- **SLOC – Source lines of code (SLOC)**
 - **Software metric that measures the amount of code in a program**
- **According to Gary McGraw:**
 - **Win 95 – 15 Millions SLOC**
 - **Win 98 – 18 Millions SLOC**
 - **Win XP – 40 Millions SLOC**

Software Size

- **Small software projects**
 - Can keep track of details in head
 - Last for short periods
 - What students learn in school
- **Large projects**
 - Much more complex
 - Commonly found in real world
 - Why we try to teach you
 - Software engineering
 - Object-oriented programming

Software Life Cycle

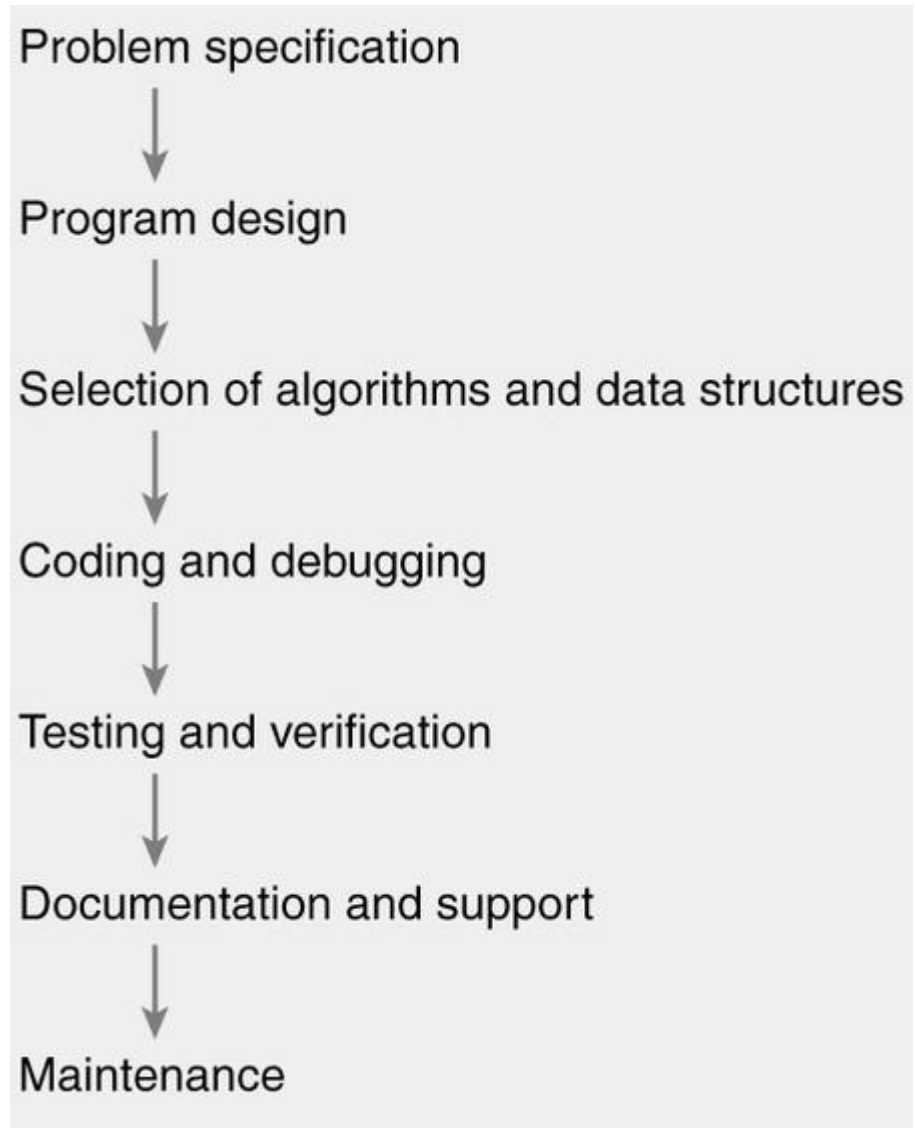
- **Coding is only part of software development**
- **Software engineering requires**
 - **Preparation before writing code**
 - **Follow-up work after coding is complete**
- **Software life cycle**
 - **Certain essential operations needed for good software development**
 - **No universal agreement, just general agreement**

Components of Software Life Cycle

- 1. Problem specification**
- 2. Program design**
- 3. Algorithms and data structures**
- 4. Coding and debugging**
- 5. Testing and verification**
- 6. Documentation and support**
- 7. Maintenance**

“Waterfall Model” of Life Cycle

- **Simple model**
 - Proceed from one step to next
 - Result of step flow into next
- **In reality**
 - May need to return to previous step
 - Steps may be more integrated
 - Steps may occur at same time



Software Life Cycle

■ Waterfall model

- Reasonable for small projects
- Unworkable for large projects

■ Some alternative approaches

■ “Unified model”

- Iteratively add incremental improvements

■ “Extreme programming”

- Write test cases **before** writing code

■ “Rapid prototyping”

- Use working prototypes to refine specifications