CS 171: Introduction to Computer Science II

Department of Mathematics and Computer Science

Li Xiong
Roadmap

• Lab session
• Pretest Postmortem
• Java Review
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
  – Arrays
  – OO and Inheritance
Lab Session

- Option 1: Monday 10-11am
- **Option 2: Monday 5-6pm (selected)**
- Option 3: Friday 10-11am

- First lab session: Eclipse and debugging lab
Debugging

- Debugging - finding and correcting errors

- Approaches
  - Hand-trace the program
  - Insert print statements to show the values of the variables or the execution flow
    ```java
    System.out.println("radius = "+ radius);
    ```
  - Use a debugger utility
Programming Errors

- Syntax Errors
  - Detected by the compiler
  - E.g. variable not initialized
- Runtime Errors
  - Causes the program to abort
  - E.g. array index out of bounds
- Logic Errors
  - Produces incorrect result
Roadmap

• Lab session
• Pretest Postmortem
• Java Review
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
  – Arrays
  – OO and Inheritance
# Pretest Postmortem

<table>
<thead>
<tr>
<th>Question</th>
<th>Topics</th>
<th>#correct answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loops; post increment operator</td>
<td>15/31</td>
</tr>
<tr>
<td>2</td>
<td>Arithmetic operations - division; modulo</td>
<td>17/31</td>
</tr>
<tr>
<td>3</td>
<td>Object variables; null references</td>
<td>2/31</td>
</tr>
<tr>
<td>4</td>
<td>Integer variables</td>
<td>23/31</td>
</tr>
<tr>
<td>5</td>
<td>Object variables</td>
<td>23/31</td>
</tr>
<tr>
<td>6</td>
<td>Loops; arrays; problem solving</td>
<td>8/31</td>
</tr>
<tr>
<td>7a)</td>
<td>Inheritance</td>
<td>28/31</td>
</tr>
<tr>
<td>7b)</td>
<td>Inheritance; class constructor</td>
<td>11/31</td>
</tr>
<tr>
<td>7c)</td>
<td>Methods; overloading; polymorphism</td>
<td>1/31</td>
</tr>
<tr>
<td>7d)</td>
<td>Inheritance; problem solving</td>
<td>7/31</td>
</tr>
</tbody>
</table>
Roadmap

• Lab session
• Pretest Postmortem
• Java Review
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
  – Arrays
  – OO and Inheritance
Data Types

• Primitive types
  – 6 numeric types
    • 4 integral types: byte, short, int, long
    • 2 floating point types: float, double
  – 1 character type: char
  – 1 boolean type: boolean

• Reference types
  – Class types, interface types, array types
  – Special null types
Variables

• A variable is a name for a location in memory used to hold a data value.
  – Type, name and contents

• Using a variable
  – Declaring a variable – type and name
    • Instructs the compiler to reserve a portion of main memory to hold a particular type of value referred by a particular name
  – Assign a value to a variable
  – Use a variable in an expression
    • A variable cannot be used if it is not declared or initialized
    • The left hand side of the assignment operator is always a variable and the right hand side is an expression
Using Variables

• Declaring a variable

```c
int count;
```

• Assign a value to a variable

```c
count = 0;
```

• Declaring and initializing in one step

```c
int count = 0;
```

• Using a variable in an expression

```c
count += 1;
count ++;
```
Primitive data types vs. object data types

- Variables of primitive data types store the actual value
- Variables of object types store the reference to the object

```
int i = 1;
Circle c = new Circle();
```
The null Value

• If a data field of a reference type does not reference any object, the data field holds a special value: null

Circle c;
double r = c.getRadius();
Question

• A variable, int x stores: ____________

A. A reference to an int
B. An integer value
C. The identifier, ”x”
D. Lots of goodies for every good Java-slave
Question

• A variable, BankAccount x stores: __________
  – A reference to an object of the BankAccount class
  – An object of the BankAccount class
  – The identifier, ”x”
  – Even more goodies than a mere int x
Question

• Which of the following will always correctly check whether an object variable obj contains a null reference? __________

A) obj.equals(null);
B) null == obj;
C) obj = null;
D) null.equals(obj);
E) None of the above
Expressions

• An expression is a combination of one or more operators and operands that perform a calculation
  – Operands might be numbers, variables, or expressions

• Arithmetic expressions

  int score = score - 10 * lateDays

• Boolean expressions

  boolean isLate = submissionDate <= dueDate;
## Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>$34 + 1$</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>$34.0 - 0.1$</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>$300 \times 30$</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>$1.0 / 2.0$</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>$20 % 3$</td>
<td>2</td>
</tr>
</tbody>
</table>

- Division performs integer division when both operands are integers
Question

• If a and b are ints such that b != 0 then which of the following expressions is always equivalent to a%b? ___________________

A) a-(a/b)*b
B) (a/b)*b
C) a-a/b
D) (double)a/b - a/b
# Shortcut Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>f -= 8.0</td>
<td>f = f - 8.0</td>
</tr>
<tr>
<td>*=</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>%=</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>
## Increment and Decrement Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++var</td>
<td>preincrement</td>
<td>The expression (++var) increments var by 1 and evaluates to the <em>new</em> value in var <em>after</em> the increment.</td>
</tr>
<tr>
<td>var++</td>
<td>postincrement</td>
<td>The expression (var++) evaluates to the <em>original</em> value in var and increments var by 1.</td>
</tr>
<tr>
<td>--var</td>
<td>predecrement</td>
<td>The expression (--var) decrements var by 1 and evaluates to the <em>new</em> value in var <em>after</em> the decrement.</td>
</tr>
<tr>
<td>var--</td>
<td>postdecrement</td>
<td>The expression (var--) evaluates to the <em>original</em> value in var and decrements var by 1.</td>
</tr>
</tbody>
</table>
Increment and Decrement Operators, cont.

```c
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```c
int i = 10;
int newNum = 10 * (i + 1);
```

```c
int i = 10;
int newNum = 10 * i;
```

Same effect as

```c
i = i + 1;
int newNum = 10 * i;
```
The Boolean Expressions

• A Boolean expression evaluates to a Boolean value

• Comparison operators: compare a pair of values (numbers, characters, boolean values)
  
  boolean happy = grade > 90;

• Boolean operators: perform logic operations (boolean values)
  
  boolean happy = (grade > 90) && (workhours < 2);
## Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
</tbody>
</table>
Comparing objects

- `==` compares references
  - Check whether an object variable contains a null reference
- `equals()` method compares contents
  - The default implementation of the equals method in the Object class:

```java
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Java classes such as String override equals() method so that it compares the content of two objects.
- It is a good idea to override equals() method for your own classes
Roadmap

• Lab session
• Pretest Postmortem
• Java Review
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
  – Arrays
  – OO and Inheritance
Simple if Statements

if (booleanExpression) {
    statement(s);
}

if (!passedTest) {
    System.out.println("I’ll write good code!");
}

(A)

(B)
The if...else Statement

```java
if (booleanExpression) {
    statement(s)-for-the-true-case;
}
else {
    statement(s)-for-the-false-case;
}
```
World Without Loops is Painful...

```java
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
System.out.println("I will write good code!");
```
A Better Approach: **Loops**

```java
int count=0;
while (count < 100){
    System.out.println("I will write good code!");
    count++;
}
```
while (loop-continuation-condition) {
    // loop-body;
    Statement(s);
}

int count = 0;
while (count < 100) {
    System.out.println("I’ll write good code!");
    count++;
}
do-while Loop

```c
    do {
        // Loop body;
        Statement(s);
    } while (loop-continuation-condition);
```
for (initial-action; loop-condition; action-after-each-iteration) {
   // loop body;
   Statement(s);
}

for (int count=0; count < 100; count++) {
   System.out.println(“I will write good code!”);
}
**for Loops**

for (initial-action; loop-continuation-condition; action-after-each-iteration) {
    // loop body;
    Statement(s);
}

for (int i = 0; i < 100; i++) {  
    System.out.println("I’ll write good code!");
}

**Flowchart Diagram**

(A) Initial-Action

Loop Continuation Condition?

true

Statement(s) (loop body)

Action-After-Each-Iteration

false

(B) i = 0

(i < 100)?

true

System.out.println("I’ll write good code");

false

i++
Which loop to use?

• Use the one that is most intuitive and comfortable for you.

• A for loop may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.

• A while loop may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.

• A do-while loop can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.
Question

• What is the output of the following code fragment? ________________.

```java
int sum = 1;
for (int i = 0; i <= 5; sum = sum + i++);
System.out.print(sum);
```
What is the value of x after the following statements? ___

```java
int x = 0, j = 0;
boolean done = false;
while(!done) {
    for (int i = 0; i<5; i++) {
        j = j + i;
        if (j > 12) {
            x = j;
            done = true;
        }
    }
}
```
Roadmap

• Lab session
• Pretest Postmortem
• Java Review
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
  – Arrays
  – OO and Inheritance
Levels of Abstraction: Software Design

• Old times: computer programs manipulated primitive types such as numbers and characters
• Methods: Encapsulate routine computations to black boxes
• Object-oriented programming: Encapsulate data fields and methods to black boxes
Example – Computing sum

```java
int sum = 0; int n=5;
for (int i = 1; i <= n; i++) {
    sum += i;
    sum += i;
}
System.out.println("sum is:" + sum);
```
public static void main(String[] args) {
    int sum = 0;
    for (int i = 1; i <= 10; i++) {
        sum += i;
    }
    System.out.println("The sum of 1-10 is: " + sum);
    sum = 0;
    for (int i = 25; i <= 30; i++) {
        sum += i;
    }
    System.out.println("The sum of 25-30 is: " + sum);
    sum = 0;
    for (int i = 40; i <= 50; i++) {
        sum += i;
    }
    System.out.println("The sum of 40-50 is: " + sum);
}
Using a method *Sum*

If there was a method *sum* that would take two input integer arguments, *start* and *end*, and would add up and *return* sum of numbers from *start* to *end*

The program could be re-written much easier as:

```java
public static void main(String[] args) {
    System.out.println("sum(1, 10) is: " + sum(1, 10)); // 1+2+...+10
    System.out.println("sum(25, 30) is: " + sum(25, 30)); //25+26+...+30
    System.out.println("sum(40, 50) is: " + sum(40, 50)); //40+41+...+50
}
```
Defining a Method \textit{sum} \\
\begin{verbatim}
public static int sum(int start, int end) {
    int sum = 0;
    for (int i = start; i <= end; i++) {
        sum += i;
    }
    return sum;  // return is required
}
\end{verbatim}
Defining and Using Methods

• Define a method – give a definition of what the method is to do

```
modifier returnType methodName(list of parameters) {
    collection of statements;
}
```

• Call or invoke a method – use a method

```
methodName(list of parameters)
```

![Diagram of method definition and invocation](image)