CS 171: Introduction to Computer Science II

Department of Mathematics and Computer Science

Li Xiong
Roadmap

• Pretest Postmortem
• Java Review/Basics
  – 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 (partially correct) answers / # total answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Loops; post increment operator</td>
<td>13/33</td>
</tr>
<tr>
<td>2</td>
<td>Arithmetic operations - division; modulo</td>
<td>18/33</td>
</tr>
<tr>
<td>3</td>
<td>Object variables; null references</td>
<td>2/33</td>
</tr>
<tr>
<td>4</td>
<td>Integer variables</td>
<td>27/33</td>
</tr>
<tr>
<td>5</td>
<td>Object variables</td>
<td>20/33</td>
</tr>
<tr>
<td>6a)</td>
<td>Inheritance</td>
<td>26/33</td>
</tr>
<tr>
<td>6b)</td>
<td>Methods; overloading; polymorphism</td>
<td>6/33</td>
</tr>
<tr>
<td>6c)</td>
<td>Inheritance; problem solving</td>
<td>14/33</td>
</tr>
<tr>
<td>7</td>
<td>Loops; arrays; problem solving</td>
<td>18/33 (14/33)</td>
</tr>
</tbody>
</table>
Data Types

• Primitive types
  – Integers, with arithmetic operations: \texttt{byte}, \texttt{short}, \texttt{int}, \texttt{long}
  – Real numbers, with arithmetic operations: \texttt{float}, \texttt{double}
  – Booleans, with \{true, false\} values and logical operations: \texttt{boolean}
  – Characters: \texttt{char}

• Reference types
  – Class types, interface types, array types
  – Special \texttt{null} types
Variables

• A variable is a name for a location in memory used to hold a data value.
  – Name (identifier), type, and value

• 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
  ```
  int i;
  ```
- Assign a value to a variable
  ```
  i = 1;
  ```
- Declaring and initializing in one step
  ```
  int i = 1;
  ```
- Using a variable in an expression
  ```
  i += 1;
  ```
  ```
  i ++;
  ```
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
  - If it does not reference any object, it holds a special value: null

```java
Circle c;
c = new Circle(1);
double r = c.getRadius();
```
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
  • If it does not reference any object, it holds a special value: null

```java
Circle c;
c = new Circle(1);
double r = c.getRadius();
```

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>int i = 1</th>
<th>i</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object type</td>
<td>Circle c</td>
<td>c</td>
<td>reference</td>
</tr>
</tbody>
</table>

Created using new Circle()

```
c: Circle
radius = 1
```
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

```java
int score = score - 10 * lateDays;
```

• Boolean expressions

```java
boolean isLate = submissionDate <= dueDate;
boolean happy = (grade > 90) && (workhours < 2);
```
<table>
<thead>
<tr>
<th>type</th>
<th>set of values</th>
<th>operators</th>
<th>typical expressions</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers between $-2^{31}$ and $+2^{31} - 1$</td>
<td>+ (add)</td>
<td>5 + 3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>(32-bit two’s complement)</td>
<td>- (subtract)</td>
<td>5 - 3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* (multiply)</td>
<td>5 * 3</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ (divide)</td>
<td>5 / 3</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>% (remainder)</td>
<td>5 % 3</td>
<td>2</td>
</tr>
<tr>
<td>double</td>
<td>double-precision real numbers</td>
<td>+ (add)</td>
<td>3.141 - 0.03</td>
<td>3.111</td>
</tr>
<tr>
<td></td>
<td>(64-bit IEEE 754 standard)</td>
<td>- (subtract)</td>
<td>2.0 - 2.0e-7</td>
<td>1.9999998</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* (multiply)</td>
<td>100 * 0.015</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/ (divide)</td>
<td>6.02e23 / 2.0</td>
<td>3.01e23</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>&amp;&amp; (and)</td>
<td>true &amp;&amp; false</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(or)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>! (not)</td>
<td>!false</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>^ (xor)</td>
<td>true ^ true</td>
<td>false</td>
</tr>
<tr>
<td>char</td>
<td>characters (16-bit)</td>
<td>[arithmetic operations, rarely used]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


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

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

Same effect as

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

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

Same effect as

```c
i = i + 1;
int newNum = 10 * i;
```
Roadmap

• Pretest Postmortem
• Java Review/Basics
  – Types, variables, assignments, expressions
  – Control flow statements
  – Methods
• Arrays
• OO and Inheritance
Statements

• Declaration
• Assignments
• Conditionals
• Loops
• Break and continue
Simple if Statements

if (booleanExpression) {
    statement(s);
}

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

(A)

(B)
The if...else Statement

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

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

do {
    // Loop body;
    Statement(s);
}
while (loop-continuation-condition);
for Loops

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

```java
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!");
}
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.
<table>
<thead>
<tr>
<th>Statement</th>
<th>Examples</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration</td>
<td>int i; double c;</td>
<td>create a variable of a specified type, named with a given identifier</td>
</tr>
<tr>
<td>assignment</td>
<td>a = b + 3; discriminant = b<em>b - 4.0</em>c;</td>
<td>assign a data-type value to a variable</td>
</tr>
<tr>
<td>initializing</td>
<td>int i = 1; double c = 3.141592625;</td>
<td>declaration that also assigns an initial value</td>
</tr>
<tr>
<td>implicit</td>
<td>i++; i += 1;</td>
<td></td>
</tr>
<tr>
<td>conditional (if)</td>
<td>if (x &lt; 0) x = -x;</td>
<td>execute a statement, depending on boolean expression</td>
</tr>
<tr>
<td>conditional (if-else)</td>
<td>if (x &gt; y) max = x; else max = y;</td>
<td>execute one or the other statement, depending on boolean expression</td>
</tr>
<tr>
<td>loop (while)</td>
<td>int v = 0; while (v &lt;= N) v = 2*v;</td>
<td>execute statement until boolean expression is false</td>
</tr>
<tr>
<td>loop (for)</td>
<td>for (int i = 1; i &lt;= N; i++) sum += 1.0/i;</td>
<td>compact version of while statement</td>
</tr>
<tr>
<td>call</td>
<td>int key = StdIn.readInt();</td>
<td>invoke other methods (see page 22)</td>
</tr>
<tr>
<td>return</td>
<td>return false;</td>
<td>return from a method (see page 24)</td>
</tr>
</tbody>
</table>
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);
```
Bonus question

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/Basics
  – 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
int sum = 0;
for (int i = 1; i <= 5; 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);
}
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}
Using a method Sum

```java
public static void main(String[] args) {
    System.out.println("sum(1, 10) is: " + sum(1, 10) );

    System.out.println("sum(25, 30) is: " + sum(25, 30) );

    System.out.println("sum(40, 50) is: " + sum(40, 50) );
}
```
Defining and Using Methods

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

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

• Call or invoke a method – use a method

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

Diagram:
- Method header:
  - Modifier
  - Return type
  - Method name
  - Formal parameters
- Method body:
  - Public static int max
  - Parameter list: int num1, int num2
  - Collection of statements:
    - if (num1 > num2)
      - result = num1;
    - else
      - result = num2;
    - return result;
- Invoke a method:
  - int z = max(x, y);
Passing Parameters

- When calling a method, the arguments must match the parameters in order, number, and compatible type.

  ```java
class Print {
    public static void nPrintln(String message, int n) {
      for (int i = 0; i < n; i++)
        System.out.println(message);
    }

    public static void main(String[] args) {
      nPrintln("Hello!", 3);
      nPrintln("So that’s how the methods work", 10);
    }
  }
```

- When invoking a method, the value of the argument is passed to the parameter. The variable itself is not affected. This is referred to as **pass-by-value**.
Mechanics of the Method-Calling Process

1. Evaluate the argument expressions
2. Copy argument value into the corresponding parameter, (allocated in a newly assigned region of memory called a stack frame)
3. Execute body, using the new stack frame for local variables.
4. On a return statement, compute the return value and substitutes that value in place of the call.
5. Discard the stack frame for the method and returns to the caller, continuing where it left off.
public static void main(String[] args) {
    int i = 0;
    ...
    // 1. evaluate arguments
    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
}

public static int sum(int start, int end) { // 2. copy args, new SF
    int sum = 0; // 3. execute the body
    for (int i = start; i <= end; i++) {
        sum += i;
    }
    return sum; //4 and 5. return the value, and discard stack frame
}
Overloading methods

• Method overloading: multiple methods can have the same name but different parameter lists

• Compiler determines which method is used based on the method signature (method name and parameters)
  – Early binding
Overloading Methods

public static int max(int num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

max(1, 3);
max(1.0, 3.0);
max(1.0, 3);
Overloading Methods

```java
public static int max(int num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

max(1, 3);
max(1.0, 3.0);
max(1.0, 3);
```
Overloading

/** a generic employee class */
public class Employee
{
    private String name; // name of the employee
    public Employee (String n) { name = n; }
    public Employee () { name = "Unknown"; }
    public String getName() { return name; }
    public String toString() { return name; }
    public double earnings() { return 0; }
}

/** An hourly employee that makes an earning based on hourly wage */
public class HourlyEmployee extends Employee
{
    private double wage;
    private double hours;
    public HourlyEmployee(String n, double w, double h) {
        super(n); wage = w; hours = h; }
    public double earnings() {
        return wage * hours; }
}

/** A salaried employee that makes a fixed salary */
public class SalariedEmployee extends Employee
{
    private double weeklySalary;
    public SalariedEmployee(String n, double salary) {
        super(n); weeklySalary = salary; }
    public double earnings() {
        return weeklySalary; }
}
Roadmap

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