CS1101 Programming and Problem Solving

Dr. Gina Bai

Spring 2023

Logistics

- **ZY-2B** on **zyBook > Assignments**
  - Due: Saturday, Jan 28, at 11:59pm
- **PA02 – W1, W2, A, B** on **zyBook > Chap 11**
  - Due: Thursday, Feb 2, at 11:59pm

Logistics – codePost

- **PA00 grade is available on codePost**
  - Email notifications as well as the codePost invitation were sent on Jan 24
  - Sign up to codePost promptly
  - Read the *Introduction to codePost Interface*
    - Brightspace | Course Documents
    - Scores, comments, deductions, remaining free late days

Logistics – Free Late Days

- No submission will be accepted **48 hours** after the deadline, regardless of the use of free late days
    - **Up to two** free late days can be applied **per assignment**
    - Free late days will be applied **automatically**

- Submission is “per PA assignment”-based, NOT “per problem”-based
    - **Submit ALL problems on time**

Recap

Q: Identify the four program errors in FavNum.java, and specify the following for each error: 1) location, 2) type, 3) how to fix

Syntax error
Missing import statement: import java.util.Scanner;

```
public class FavNum {
    public static void main(String[] args) {

        Scanner console = new Scanner(System);

        fav = console.next();

        System.out.println("What is your favorite number?");

        System.out.println("Your favorite number is " + fav + ".");
    }
}
```

Syntax error
System.in

Syntax error
Not declared

Logic error
Swap the order of these two statements

Recap

Q: Evaluate the following expressions

`Math.abs(-33)`	`33`
`Math.pow(4, 2)`	`16.0`
`Math.floor(-6.7)`	`-7.0`
`Math.ceil(-6.7)`	`-6.0`
`Math.round(-6.7)`	`-7.0`

Specialized Assignment Operators (Shorthands)

Shortcut Operators

Standard expression	**Equivalent shorthand version**
varName = varName + <expression>;	varName **+=** <expression>;
varName = varName – <expression>;	varName **–=** <expression>;
varName = varName ∗ <expression>;	varName **∗=** <expression>;
varName = varName / <expression>;	varName **/=** <expression>;
varName = varName % <expression>;	varName **%=** <expression>;

Q: What’s the exact output of the following code?

```java
int x = 12;
x /= 12;
System.out.println(x);
```

1

```java
final int MAX = 5;
int x = 12;
x -= MAX;
System.out.println(x);
```

7

Increment ++ and Decrement Operators --

Increment by 1	Decrement by 1
`varName = varName + 1;`	`varName = varName - 1;`
`varName += 1;`	`varName -= 1;`
`varName++;`	`varName--;`
`++varName;`	`--varName;`

Prefix (++var or --var)

- Step 1: Increment/Decrement the value of var
- Step 2: Use the updated value of var in the statement

```java
class PrefixDemo {
    public static void main(String[] args) {
        int a = 5;
        System.out.println("a is " + a);
        int b = ++a;
        System.out.println("a is " + a);
        System.out.println("b is " + b);
    }
}
```

```
$ javac PrefixDemo.java
$ java PrefixDemo
a is 5
a is 6
b is 6
```

Equivalent to →

```java
a += 1;
int b = a;
```

Postfix (var++ or var--)

- Step 1: Use the current value of var in the statement
- Step 2: Increment/Decrement the value of var

```java
class PostfixDemo {
    public static void main(String[] args) {
        int a = 5;
        System.out.println("a is " + a);
        int b = a++;
        System.out.println("a is " + a);
        System.out.println("b is " + b);
    }
}
```

```
$ javac PostfixDemo.java
$ java PostfixDemo
a is 5
a is 6
b is 5
```

Equivalent to →

```java
int b = a;
a += 1;
```

Q: What’s the exact output of the following code?

```
int a = 2;
int b = 2 * (++a);
System.out.println("a is " + a);
System.out.println("b is " + b);
```

```
a is 3
b is 6
```

```
int a = 2;
int b = 2 * (a++);
System.out.println("a is " + a);
System.out.println("b is " + b);
```

```
a is 3
b is 4
```

Static Methods
Parameters & Return Values

zyBook Chap 3.1, 3.2, 3.3, 3.4

```java
public class PrintFace {
    public static void main(String[] args) {
        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |  /\\  /\\   | |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");

        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |   X   X   | |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");
    }
}
```

```
$ javac PrintFace.java
$ java PrintFace
  _______________ 
 |  ___________  |
 | |           | |
 | |  /\  /\   | |
 | |    ___    | |
 | |___________| |
 |_______________|
    _|_______|_


  _______________ 
 |  ___________  |
 | |           | |
 | |   X   X   | |
 | |    ___    | |
 | |___________| |
 |_______________|
    _|_______|_
```

```java
public class PrintFace {
    public static void main(String[] args) {
        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |  /\\  /\\   |  |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");

        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |   X   X   | |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");
    }
}
```

Can this code be improved?
Any repetition of the code?

```java
public class PrintFace {
    public static void main(String[] args) {
        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |  /\\  /\\   | |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");

        System.out.println("  _______________ ");
        System.out.println(" |  ___________  |");
        System.out.println(" | |           | |");
        System.out.println(" | |   X   X   | |");
        System.out.println(" | |    ___    | |");
        System.out.println(" | |___________| |");
        System.out.println(" |_______________|");
        System.out.println("    _|_______|_\n");
    }
}
```

// Header
// Happy eyes
// Footer

// Header
// Unhappy eyes
// Footer

Can this code be improved?
Any repetition of the code?

Methods

A group of statements with a given name

- Decompose a program into smaller modules that
  - Each module implements a part of the program behavior, and
  - Can be implemented and tested separately
- Eliminates redundancy by allowing code reuse

Methods

Equivalent Implementations

```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
        System.out.println("Have a great day!");
    }
}
```

```java
public class HelloWorld {

    public static void main(String[] args) {
        printMessage();
    }

    public static void printMessage() {
        System.out.println("Hello World!");
        System.out.println("Have a great day!");
    }
}
```

<return type> <methodName> (<parameter(s)>)

Step 2: Call the method
Inside of the main method

Step 1: Declare the method
Inside of the class, outside of the main method

Return Type and Return Statement

The **type** of the **output** generated by the method, if any

- Could be an int, a double, a char, a boolean, or a String, …
- A method can generate only **ONE** value, that is, return one value
- If a method does not generate a value, the return type is **void**
    - E.g., contains print statements only

```
// No parameters; Has a return value
public static <type> <methodName>() {
    <statements>;
    return <expression>;
}


// No parameters; Has no return value
public static void <methodName>() {
    <statements>;
}
```

Parameters of a Method

The **input** into the method

- Each method can have 0, 1, or many parameters.
- Each parameter has a **type** and **name** (similar to variables).
- The **scope** of parameters is the method.

```
// Two parameters; Has a return value
public static <type> <methodName>(<type> < paraName>, <type> < paraName>) {
    <statements>;
    return <expression>;
}
```

Flow of Control

- Flow of Control is the order that statements execute.
- With methods:
  - Control is transferred to the called method
  - When the called method is complete, the control returns to the calling method

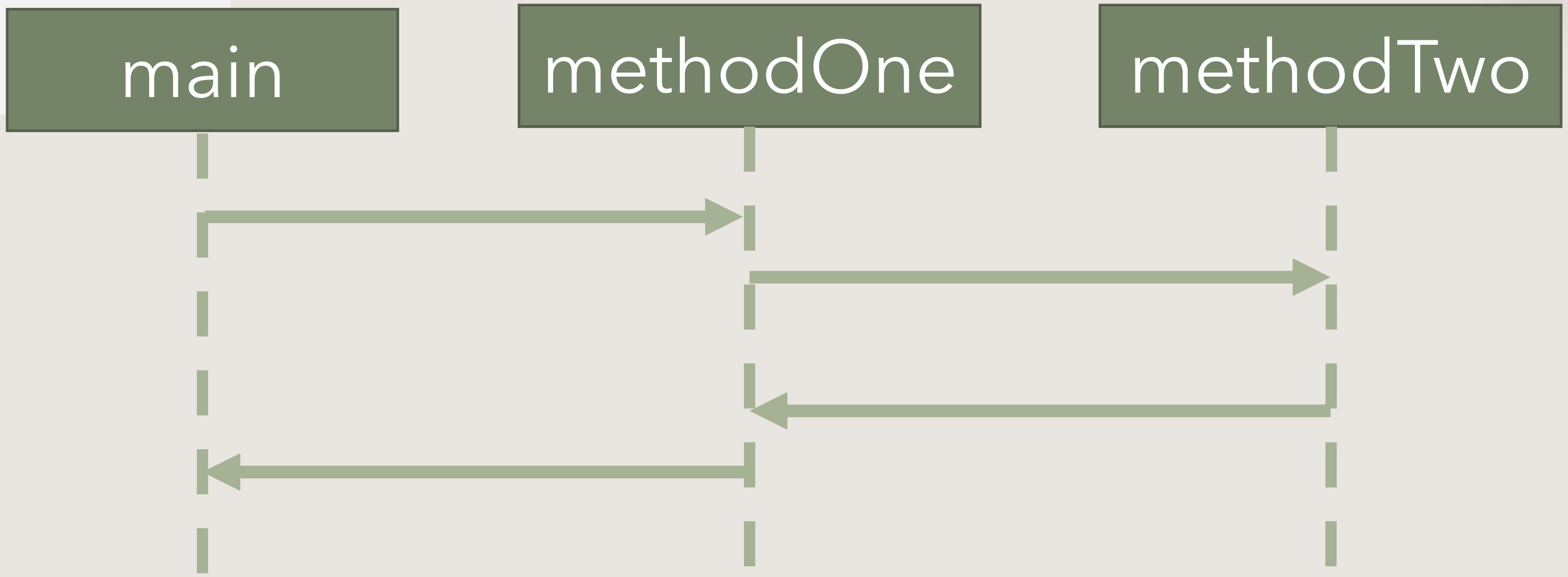
```
public class FlowOfControlDemo {
    public static void main(String[] args) {
        methodOne();
    }

    public static void methodOne() {
        System.out.println("A");
        System.out.println("B");
        methodTwo();
        System.out.println("C");
    }

    public static void methodTwo() {
        System.out.println("X");
        System.out.println("Y");
        System.out.println("Z");
    }
}
```

Output:

A
B
X
Y
Z
C



Q: What's the exact output of the following code?

```java
public class MethodExample {

    public static void main(String[] args) {
        m1(4);
        int x = m2(2, 4);
        System.out.println("x is " + x + ".");
    }

    public static void m1(int x) {
        System.out.println("m1 prints its parameter " + x + ".");
    }

    public static int m2(int a, int b) {
        System.out.println("m2 is called.");
        return a + b;
    }
}
```

Since m2 generates an integer result, we declare an integer x to hold its return value

```
$ javac MethodExample.java
$ java MethodExample
m1 prints its parameter 4.
m2 is called.
x is 6.
```

Javadoc a Method

```java
public class MethodExample {

    public static void main(String[] args){
        m1(4);
        int x = m2(2, 4);
        System.out.println("x is " + x + ".");
    }

    /**
     * This method takes one parameter and prints it out.
     * @param  x  a value to be printed
     */
    public static void m1(int x) {
        System.out.println("m1 prints its parameter " + x);
    }

    /**
     * This method adds up its two parameters
     * @param  a  the first value to be added
     * @param  b  the second value to be added
     * @return  the sum of a and b
     */
    public static int m2(int a, int b) {
        System.out.println("m2 is called.");
        return a + b;
    }
}
```

- Description of method
- If it takes parameters, use one **@param** tag for **each** parameter. List parameter **name** followed by the **description** of the parameter.
- If it returns a value, use **@return** tag followed by the description of the return value.