Announcements

• PS 3 is due Thursday, 10/6

• Midterm Exam 1: 10/14 (Fri), 9:00am-10:53am
  • Room TBD
  • Scope: Lecture 1 to Lecture 9 (Chapters 1 to 6 of text)
  • You may bring a sheet of paper (A4, both sides)

• Tutoring schedule is now updated

• Reading assignment for this lecture: Chapter 6
Methods
Opening problem

Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.
Problem

```java
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum = sum + i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum = sum + i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum = sum + i;
System.out.println("Sum from 35 to 45 is " + sum);
```
Problem

```java
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```
Solution

```java
public static int sum(int i1, int i2) {
    int sum = 0;
    for (int i = i1; i <= i2; i++)
        sum += i;
    return sum;
}
```

```java
public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 30 is " + sum(20, 30));
    System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
```
Defining methods

A method is a collection of statements that are grouped together to perform an operation.

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

After a method is defined, it can be called (invoked).

```
int z = max(x, y);
```
Defining and calling a method

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

```
int z = max(x, y);
```
Method signature

Method signature is the combination of the method name and the parameter list.

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

int z = max(x, y);

invoke a method

actual parameters (arguments)
Formal parameters

The variables defined in the method header are known as *formal parameters*.
Actual parameters

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter* or *actual argument*.

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

```
int z = max(x, y);
```
Return value type

A method may return a value. The `returnValueType` is the data type of the value the method returns. If the method does not return a value, the `returnValueType` is the keyword `void`. For example, the `returnValueType` in the `main` method is `void`.

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

```java
int z = max(x, y);
```
Calling methods

After max is defined as a method, we can call it as in:

\[ \text{max}(i,j) \]
Calling methods

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}
```

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
Trace method invocation

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}
```

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

i is now 5
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

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

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + ", " + j + " is " + k);
}

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

(num1 > num2) is true since num1 is 5 and num2 is 2
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+
    " and "+j+" is "+k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

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

A return statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it is possible that this method does not return any value.

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return -1;
}
```

To fix this problem, delete `if (n < 0)` in (a), so that the compiler will see a return statement to be reached regardless of how the if statement is evaluated.

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else
        return -1;
}
```
Reuse methods from other classes

NOTE: One of the benefits of methods is for reuse.

Assume that max is defined in the class named TestMax.

The max method can be invoked from any class besides TestMax.

If you create a new class Test, you can invoke the max method using ClassName.methodName (e.g., TestMax.max) inside Test.
Call stacks

(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

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

j is declared and initialized

The main method is invoked.
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
Trace call stack

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

pass the values of i and j to num1 and num2

```
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}
```

The max method is invoked.

Space required for the main method

k: 5
j: 2
num1: 5
num2: 2
i: 5
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

(num1 > num2) is true

Space required for the main method
result: num2: 2
num1: 5

The max method is invoked.
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
void method example

This type of method does not return a value.

The method performs some actions though.
Passing parameters

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

Suppose you invoke the method using
    nPrintln(“Welcome to Java”, 5);
What is the output?

Suppose you invoke the method using
    nPrintln(“Computer Science”, 15);
What is the output?

Can you invoke the method using
    nPrintln(15, “Computer Science”);
Pass by value (next)

This program demonstrates passing values to the methods.

```java
public static void main(String[] args) {
    int i = 3;
    int j = 4;
    swap(i, j);
    System.out.println(i + " " + j);
}

public static void swap(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
```
Pass by value (cont.)

The values of num1 and num2 are passed to n1 and n2.

The values for n1 and n2 are swapped, but it does not affect num1 and num2.

1. The main method is invoked.
   - Activation record for the main method
     - num2: 2
     - num1: 1

2. The swap method is invoked.
   - Activation record for the swap method
     - temp: 1
     - n2: 2
     - n1: 1

3. The swap method is executed.
   - Activation record for the main method
     - num2: 2
     - num1: 1

4. The swap method is finished.
   - Activation record for the main method
     - num2: 2
     - num1: 1

5. The main method is finished.
   - Stack is empty
Modularizing code

Methods can be used to reduce redundant coding and enable code reuse.

Methods can also be used to modularize code and improve the quality of the program.

Compare Palindrome.java and Palindrome2.java
Overloading methods

- Same method name with different signatures
- Example: Overloading the name `max` with different meaning

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

Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.

This is referred to as ambiguous invocation. Ambiguous invocation is a compile error.
Example: Ambiguous invocation

```java
public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }
    public static double max(int num1, double num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
    public static double max(double num1, int num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}
```
Scope of local variables

• A *local variable*: a variable defined inside a method

• *Scope*: the part of the program where the variable can be referenced

• The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable

• A local variable must be declared before it can be used like any other variable!
Scope of local variables (cont.)

• You can declare a local variable with the same name multiple times in different non-nesting blocks in a method,

• but you cannot declare a local variable twice in nested blocks.
Scope of local variables (cont.)

• A variable declared in the initial action part of a `for` loop header has its scope in the entire loop.
• But a variable declared inside a `for` loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```java
public static void method1() {
    .
    .
    for (int i = 1; i < 10; i++) {
        .
        int j;
        .
        .
    }
}
```

The scope of `i`

The scope of `j`
Scope of local variables (cont.)

It is **fine** to declare \( i \) in two non-nesting blocks

```java
public static void method1() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

It is **wrong** to declare \( i \) in two nesting blocks

```java
public static void method2() {
    int i = 1;
    int sum = 0;
    for (int i = 1; i < 10; i++) {
        sum += i;
    }
}
```
Scope of local variables (cont.)

// Fine with no errors
public static void correctMethod() {
    int x = 1;
    int y = 1;

    // i is declared
    for (int i = 1; i < 10; i++) {
        x = x + i;
    }

    // i is declared again
    for (int i = 1; i < 10; i++) {
        y = y + i;
    }
}
Scope of local variables (cont.)

// With errors
public static void incorrectMethod() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        int x = 0;
        x = x + i;
    }
}

Method abstraction

You can think of the method body as a black box that contains the detailed implementation for the method.
Benefits of methods

• Write a method once and reuse it anywhere
• Information hiding. Hide the implementation from the user
• Reduce complexity