Method Invocation

- Note that the input parameters are sort of variables declared within the method as placeholders.
- When calling the method, one needs to provide arguments, which must match the parameters in order, number, and compatible type, as defined in the method signature.
- In Java, method invocation uses **pass-by-value**.
- When the callee is invoked, the **program control** is transferred from the caller to the callee.
- For each invocation of methods, OS creates an **frame** which stores necessary information, and the frame is pushed in the **call stack**.
- The callee transfers the program control back to the caller once the callee finishes its job.
(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.
Variable Scope

- The variable scope is the region where the variable can be referenced in the program.
- Variables can be declared in class level, method level, and loop level.
- In general, a balanced curly brackets defines a particular scope.
- One cannot declare the variables with the same name in the same scope.
Example

```java
public class ScopeDemo {

    static int i = 1; // class level

    public static void main(String[] args) {
        System.out.printf("%d\n", i); // output 1
        int i = 2; // method level; local
        i++;
        System.out.printf("%d\n", i); // output 3
        p();
        System.out.printf("%d\n", i); // output ?
    }

    static void p() {
        i = i + 1;
        System.out.printf("%d\n", i); // output ?
    }
}
```

- What if `p(int i)` in Line 14?\(^1\)

\(^1\)Thanks to a lively discussion on January 20, 2017.
A Math Toolbox: Math Class

- The Math class provides basic mathematical functions and 2 global constants Math.PI\(^2\) and Math.E\(^3\).
- All methods are public and static.
  - For example, max, min, round, ceil, floor, abs, pow, exp, sqrt, cbrt, log, log10, sin, cos, asin, acos, and random.
- Full document for Math class can be found here.
- You are expected to read the document!

---

\(^2\)The constant \(\pi\) is a mathematical constant, the ratio of a circle’s circumference to its diameter, commonly approximated as 3.141593.

\(^3\)The constant \(e\) is the base of the natural logarithm. It is approximately equal to 2.71828.
Method Overloading

- Methods with the same name can coexist and be identified by the method signatures.

```java
static int max(int x, int y) { ... }
// different numbers of inputs
static int max(int x, int y, int z) { ... }
// different types
static double max(double x, double y) { ... }
```

Recursion

Recursion is the process of defining something in terms of itself.

- A method that calls itself is said to be recursive.
- Recursion is an alternative form of program control.
- It is repetition without any loop.

Recursion is a common pattern in nature.
• Try Fractal.
Example

The **factorial** of a non-negative integer $n$, denoted by $n!$, is the product of all positive integers less than and equal to $n$.

- Note that $0! = 1$.
- For example,

$$4! = 4 \times 3 \times 2 \times 1$$
$$= 4 \times 3!$$
$$= 24.$$

- Can you find the pattern?
  - $n! = n \times (n - 1)!$
  - In general, $f(n) = n \times f(n - 1)$. 
Write a program which determines \( n! \).

```java
... static int factorial(int n) {
    if (n > 0)
        return n * factorial(n - 1);
    else
        return 1; // base case
}
...```

- Note that there must be a **base case** in recursion.
- Time complexity: \( O(n) \)
- Can you implement the same method by using a loop?
Equivalence: Loop Version

```java
...  
int s = 1;
for (int i = 2; i < n; i++) {
    s *= i;
}
...  
```

- Time complexity: $O(n)$
- One intriguing question is, Can we always turn a recursive method into a loop version of that?
- Yes, theoretically.\(^5\)

\(^5\)The Church-Turing Thesis proves it if the memory serves.
Remarks

- Recursion bears substantial overhead.
- So the recursive algorithm may execute a bit more slowly than the iterative equivalent.
- Additionally, a deeply recursive method depletes the call stack, which is limited, and causes stack overflow soon.
Memory Layout

Memory

Stack

2^{32} - 1

Heap

BSS (uninitialized)

Data (initialized)

Text (Code)

0
Example: Fibonacci Numbers

Write a program which determines $F_n$, the $(n + 1)$-th Fibonacci number.

- The first 10 Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, and 34.
- The sequence of Fibonacci numbers can be defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2},$$

where $n \geq 2$ and $F_0 = 0$, $F_1 = 1$. 
This recursive implementation is straightforward.

Yet, this algorithm isn’t efficient since it requires more time and memory.

Time complexity: $O(2^n)$ (Why?!)
So it can be done in $O(n)$ time.
It implies that the recursive one is not optimal.
Could you find a linear recursion for Fibonacci numbers?
You may try more examples.\(^6\)

\(^6\)See http://introcs.cs.princeton.edu/java/23recursion/.
Divide and Conquer

- For program development, we use the divide-and-conquer strategy\(^7\) to decompose the original problem into subproblems, which are more manageable.
  - For example, selection sort.

- Pros: easier to write, reuse, debug, modify, maintain, and also better facilitating teamwork

\(^7\)Aka stepwise refinement.
Computational Thinking

• To think about computing, we need to be attuned to three fields: science, technology, and society.
• Computational thinking shares with
  • mathematical thinking: the way to solve problems
  • engineering thinking: the way to design and evaluating a large, complex system
  • scientific thinking: the way to understand computability, intelligence, the mind and human behavior.

---

You should read this:
http://rsta.royalsocietypublishing.org/content/366/1881/3717.full
Computational Thinking Is Everywhere!

• The essence of computational thinking is abstraction.
  • An algorithm is an abstraction of a step-by-step procedure for taking input and producing some desired output.
  • A programming language is an abstraction of a set of strings each of which when interpreted effects some computation.
  • And more.

• The abstraction process, which is to decide what details we need to highlight and what details we can ignore, underlies computational thinking.

• The abstraction process also introduces layers.

• Well-defined interfaces between layers enable us to build large, complex systems.
Example: Abstraction of Computer System

Software

Application Programs

Libraries

Operating System

Drivers  Memory Manager  Scheduler

Execution Hardware

System Interconnect (bus)

Memory Translation

Controllers

I/O devices and Networking

Hardware

Controllers

Main Memory
Example: Methods as Control Abstraction

Optional arguments for input

Optional return value

Method Header

Method Body

Black box
Abstraction (Concluded)

• Control abstraction is the abstraction of actions while data abstraction is that of data structures.

• One can view the notion of an object as a way to combine abstractions of data and code.
class Lecture7 {

   // Objects and Classes

}

// Key words:

class, new, this, static, null, extends, super, abstract, final, interface, implements, protected
Observations for Real Objects

• Look around.
• We can easily find many examples for real-world objects.
  • For example, a person and his/her bottle of water.
• Real-world objects all have states and behaviors.
  • What possible states can the object be in?
  • What possible behaviors can the object perform on the states?
• Identifying these states and behaviors for real-world objects is a great way to begin thinking in object-oriented programming.
• From now, OO is a shorthand for “object-oriented.”
Software Objects

• An object keeps its states in fields and exposes its behaviors through methods.

• Plus, internal states are hidden and the interactions to the object are only performed through an object’s methods.

• This is so-call encapsulation, which is one of OO features.

• Note that the other OO features are inheritance and polymorphism, which we will see later.
• We often find many individual objects all of the same kind.
  • For example, each bicycle was built from the same blueprint so that each contains the same components.

• In OO terms, we say that your bicycle is an instance of the class of objects known as Bicycle.

• A class is the blueprint to create class instances which are runtime objects.

• Classes are the building blocks of Java applications.
Example: Points in 2D Coordinate

class Point {
    // data members: fields or attributes
    double x, y;
}

class PointDemo {
    public static void main(String[] args) {
        // now create a new instance of Point
        Point p1 = new Point();
        p1.x = 1;
        p1.y = 2;
        System.out.printf("(%d, %d)\n", p1.x, p1.y);

        // create another instance of Point
        Point p2 = new Point();
        p2.x = 3;
        p2.y = 4;
        System.out.printf("(%d, %d)\n", p2.x, p2.y);
    }
}
Class Definition

- First, give a class name with the first letter capitalized, by convention.
- The class body, surrounded by balanced braces {}, contains data members (fields) and function members (methods) for objects.