SE 102 Abstract Data Type and Problem Solving
Lecture 2

Introduction to Abstract Data Type

Assist Prof. Pree Thiengburanathum, PhD.

CAMT
College of Arts, Media and Technology
Chiang Mai University

Agenda

- Review method and array topics (continue from last week)
- Introduction to ADT
- Reading input (I/O) in JAVA

Submission Guideline

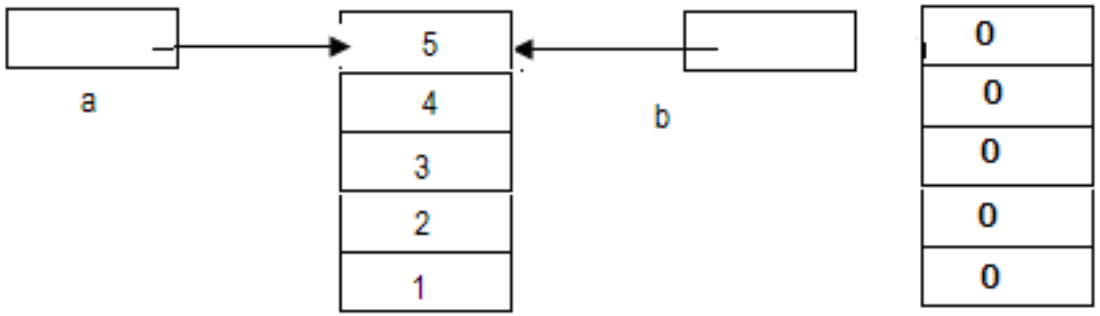
- What you need to submit
- Source code ->*.java
- Document -> README file
- Don’t need to include .class

README file (cont.)

- README.txt
- Your name, last name, student id
- How to compile your program
- How to run
- Example of output.

The = operator

- b=a



- After the assignment b=a

Abstraction

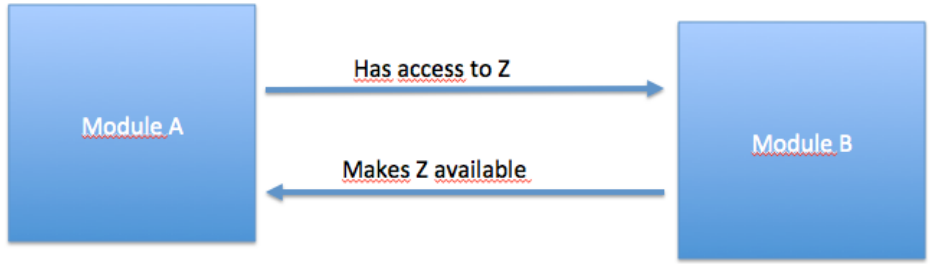
- The term '*Abstract*' in Software does not mean *vague*, *undefined*, *difficult* *to understand*

Abstraction

- "*Creating an effective usable representation of a problem by preserving the necessary elements, while eliminating unnecessary detail*"

Abstraction in Programming

- Modularity
- Keeps the complexity of a large program manageable by systematically controlling the interaction of its components
- Focus on one task at a time
- Isolates errors
- Eliminates redundancies




```
module A {
  requires B;
}
```

```
module B {
  exports Z;
}
```

Abstraction in Programming

- Each module will know what other modules do, but it does not know how other modules do the tasks
- Isolate the implementation details of a module from other module
- This is a part of a concept " *information hiding*"
- Information hiding also control the implementation detail inaccessible from outside the module

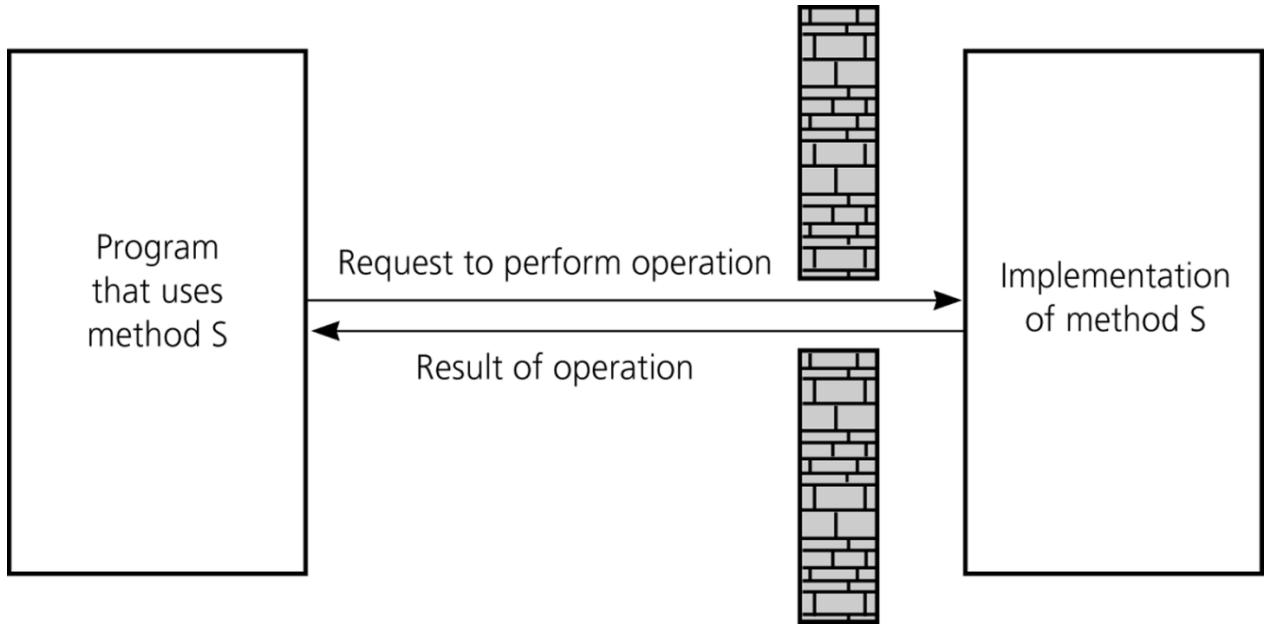
Data type

- A *data type* is a set of values together with an associated collection of operators for manipulating those values.
- 2 types of data type
  - Primitive data type (e.g., int, double, short, etc.)
  - Defined data type (e.g., array, linked list, etc.) **--> ADT**

Primitive data type vs non-primitive data type

	Primitive	Classic defined data type	Typical defined data type
	int a1;	int[] a =new int[5]	simplelist a1 =new simplelist()
Data (:=attribute)	integer	integer	Can be integer (depending on definition)
Operations(:=method)	+,-,*,/,%,!,^ &&,||,=	+,-,*,/,%,!,^ &&,||,= • Clone() • Length() • hashCode() • etc	<if it is integer> +,-,*,/,%,!,^ &&,||,= • maxVal() • minVal • Average() • insertSort() • binarySearch() • etc

Abstraction and ADT



When applied to ADT, users only need to know how to call certain ADT to store and manipulate the data. They will not know how the ADT is built

Java Method

- Through “ . ”(dot) i.e. Object (class).method
- Not only a method , fields or properties can be accessed as well
- A method or property can be accessed from other class only it permit to do so

Java Method

```
public class Caller {

public static void main(String[] args) {

      int ans=0;
      sub1.method1(1,2);
      ans=sub1.method2(5, 5);
      System.out.println("multiplication results= "+ans);
   }
        static class sub1 {

            static void  method1 (int a, int b)
            {
               System.out.println("First number is " +a+" Second number is "+b);
            }

            static int method2 (int x, int y)
            {
            return x*y;
            }


}
}
```

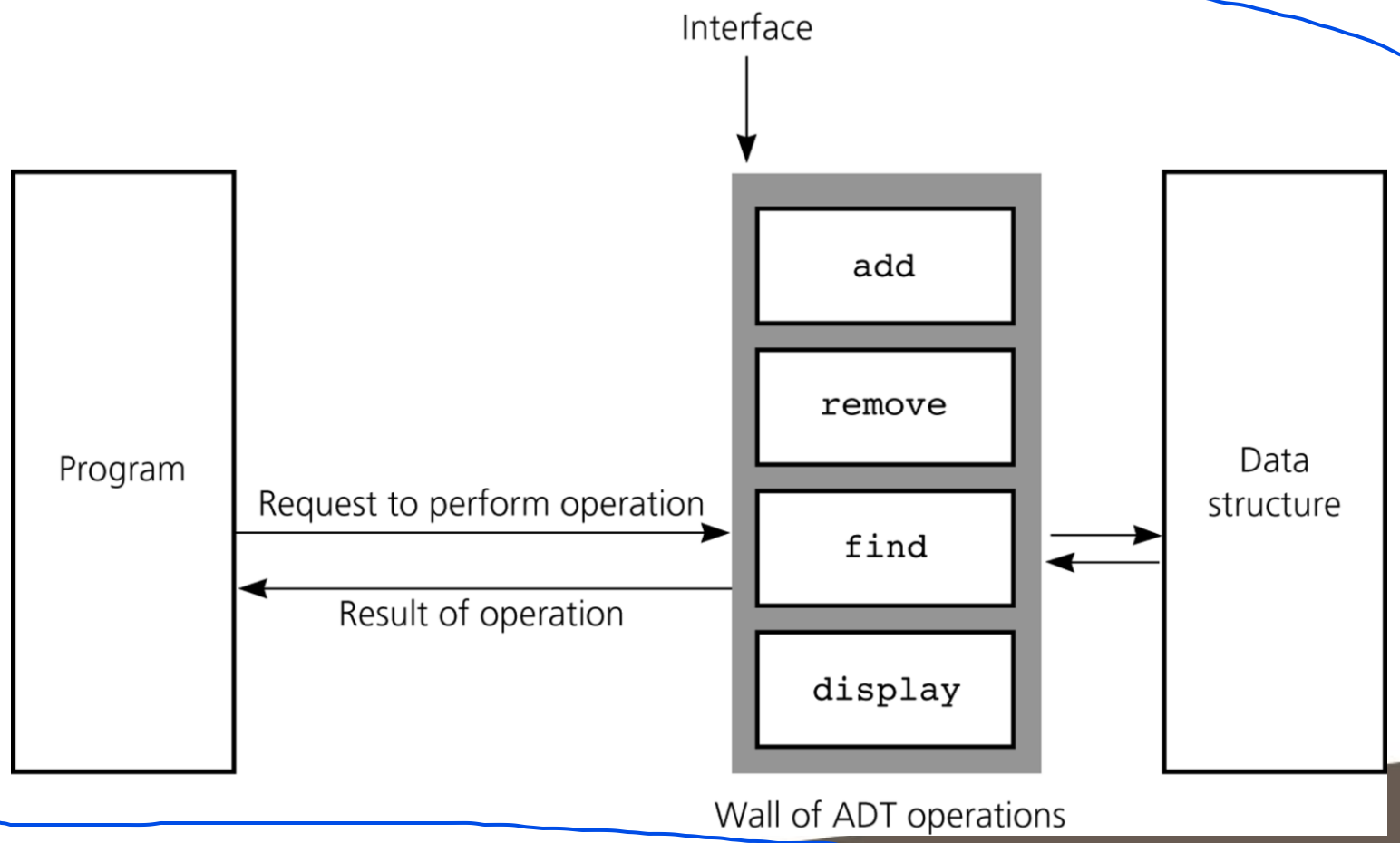
Java Method

- There are 2 classes (at this stage you can think of them as modules) : 1) Caller 2) sub1
- Caller ask sub1 do 2 tasks without any knowledge that how sub1 implement them , the class Caller only need to know what method it need to call from sub1
- Sub1 hide all implementation details, it just exposes available services to the others

What about Data Structure

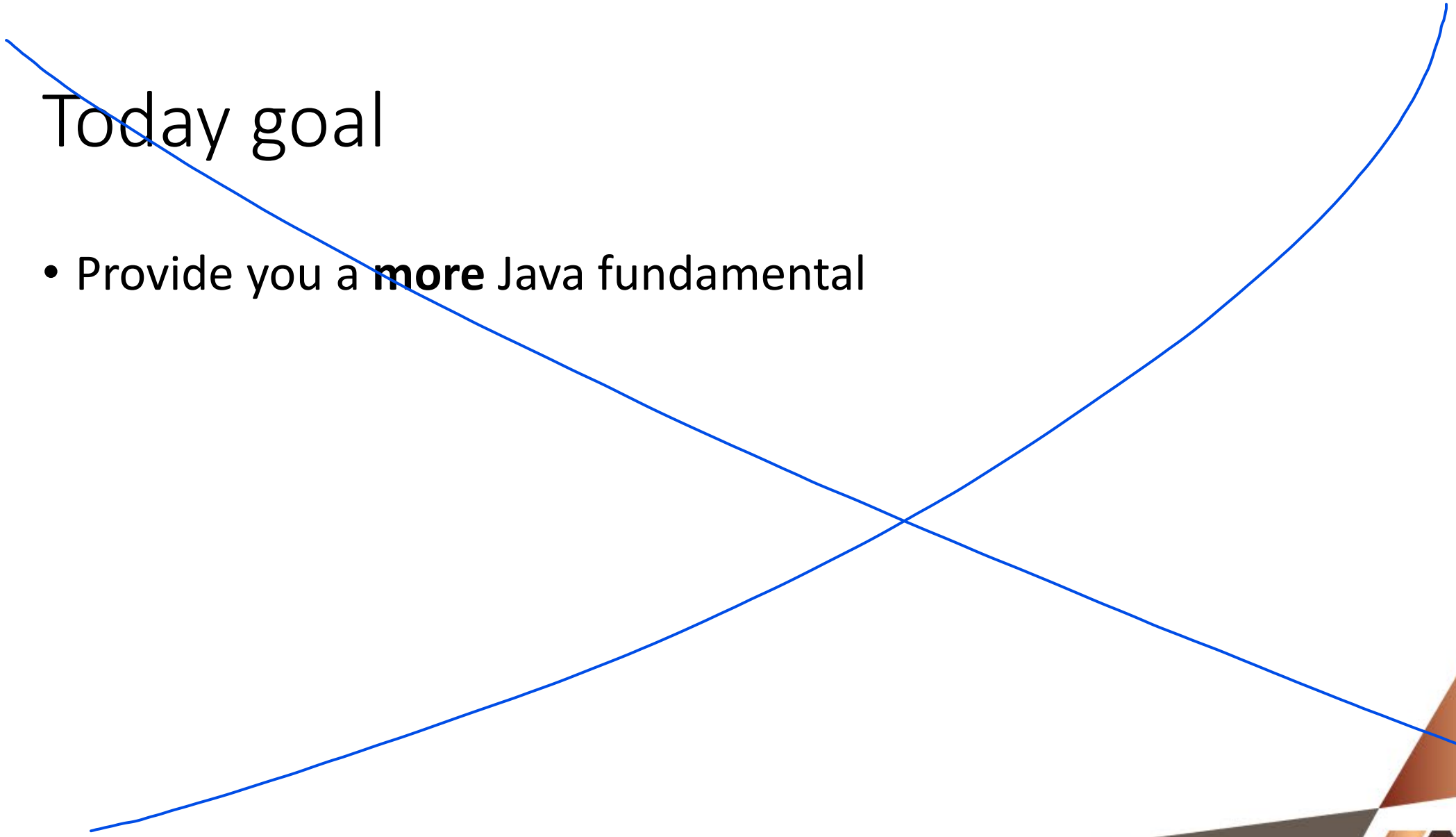
- Data structure
  - A construct that is defined within a programming language to store a collection of data
- Data abstraction
  - Results in a wall of ADT operations between data structures and the program that accesses the data within these data structures

Abstract Data Type



Today goal

- Provide you a **more** Java fundamental



Object and class

- A class is a blueprint
- An object is an instance created from that blueprint
- All objects of the same class have the same set of attributes
  - Every person object have name, weight, height
- But different value for those attributes
  - p.name = pree, p.name = john

Structure of a Java class

- Blue print of an object
- Contains attributes and behaviors
- At run time, it will create instance of the object

Syntax of class:

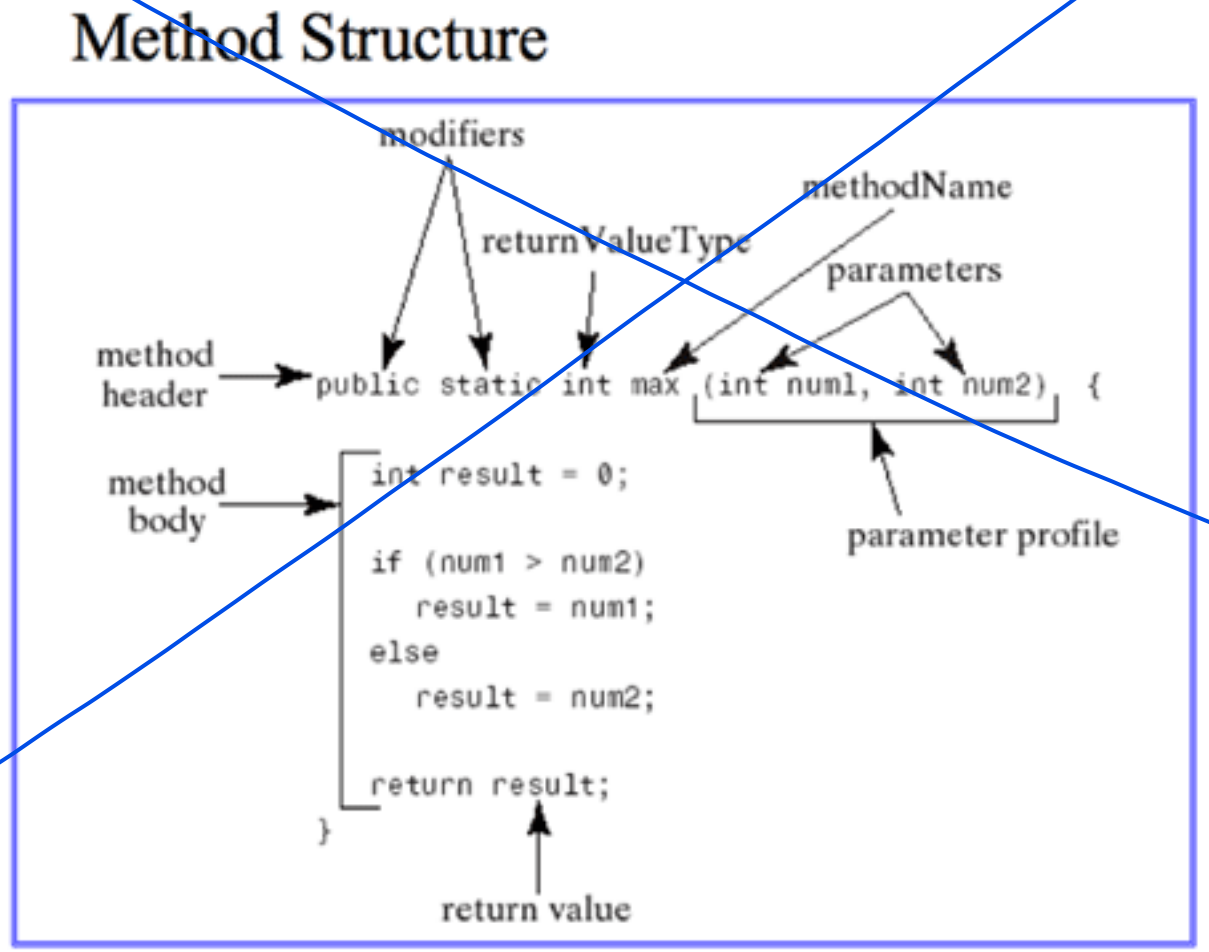
```
class classname
{       type instance-variable;
        type methodname1(parameter-list)
        {
                // body of method
        }
        type methodname2(parameter-list)
        {
                // body of method
        }
```

Java basics- method

- **Parameters**
  - Parameter list may be empty
  - Parameter list consists of comma separated pairs of types and parameter names.

```
Public void setAge(String name, int age) {
}
```

Java basics- method

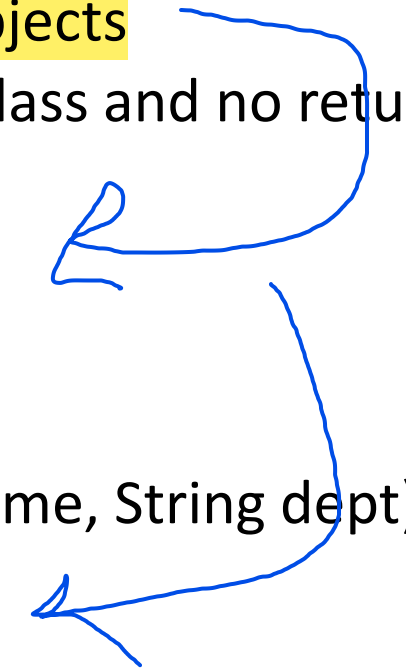


Java basics- methods

- **Constructors**
- Used to initialize new objects
- Has the same name as class and no return type

```
public counter() {
 count =0;
}

Public Professor (String name, String dept) {
name = name;
dept = dept;
}
```



Java basics- methods

- Block and Local variables
- Body of method is called block
  - A sequence of statements and declarations enclosed in branches ({});
  - Blocks may have blocks nestes
  - Variables declared with a block are known only in that block
  - These variable are called local variables

```
public static int sumThree(int a, int b, int c){
   int sum;
   int partsum = a + b;
   sum = partsum + c;
   return sum;
}
```

Java basic – control flow

If Statement

```
if (boolean_exp) {
    what_to_do_if_true
}
if (boolean_exp) {
    what_to_do_if_true
}
else {
    what_to_do_if_false
}
if (1st_boolean_exp) {
    what_to_do_if_1st_true
}
else if (2nd_boolean_exp){
    what_to_do_if_2nd_true
}
else {
    what_to_do_if_all_false
}
```

Java basic – control flow

Switch Statement

```
switch (int_type_exp) {
    case CONST1:
      action_for_CONST1;
      break;
    case CONST1:
      action_for_CONST1;
      break;
    case CONST2:
      action_for_CONST2;
      break;
    case CONST3:
      action_for_CONST3;
      break;
     . . .
    default:
      action_for_no_match;
      break;
}
```

Java basic – control flow

Switch Statement Example

```
switch (stars) {
  case 4:
    message = "truly exceptional";
    break;
  case 3:
    message = "quite good";
    break;
  case 2:
    message = "fair";
    break;
  case 1:
  case 0:
    message = "forget it";
    break;
  default:
    message = "no info found";
    break;
}
```

Java basic – control flow

While Loops

- Syntax

```
initialize
while (boolean_exp) {
    work_to_be_done
    update
}
```

- Example

```
int counter = 10;
while (counter > 0) {
    System.out.println(counter);
    counter--;
}
System.out.println("Blast Off!");
```

- What is the output?
- What if we exchange order of two statements in loop?

Java basic – control flow

For Loops

- Syntax

```
for (intialization; boolean_exp; update) {
    work_to_be_done
}
```

- Example

```
for (int counter = 10; counter > 0; counter--) {
    System.out.println(counter);
}
System.out.println("Blast Off!");
```

- What is the output?
- When is update performed?
- What is value of `counter` after loop?

Java basic – control flow

Do-While Loops

- Syntax

```
initialize
do
{
    work_to_be_done
    update
} while (boolean_exp);
```

  - NOTE REQUIRED SEMICOLON!!!

- Example

```
int counter = 10;
do {
    System.out.println(counter);
    counter-- ;
} while (counter > 0);
System.out.println("Blast Off!");
```

Java basics - loops

- Which kind of loop do we use?
- While loop
  - Don’t know of often its going to be
  - Update could be anywhere in the loop
- For loop
  - Know how often in advance
  - All information controlling loop together, in front

Tokenize

- Break a string into tokens
- Java has a class which is called “StringTokenizer”
- The StringTokenzier class helps splitting Strings in to multiple tokens.

Sample data file

10,2,4,7

3,3,1

1,3,7

2

StringTokenzier (cont.)

```
public List<String> getTokens(String str) {
    List<String> tokens = new ArrayList<>();
    StringTokenizer tokenizer = new StringTokenizer(str, ",");
    while (tokenizer.hasMoreElements()) {
        tokens.add(tokenizer.nextToken());
    }
    return tokens;
```

Java – Reading input from console

```
import java.util.Scanner;



Scanner keyboard = new Scanner (System.in);
System.out.println("Enter an integer");
int myInt = keyboard.nextInt();
```

Java – Reading input from console – cont.

```
import java.io.BufferedReader;
import java.io.InputStreamReader;


BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
String s = br.readLine();
int i = Integer.parseInt(br.readLine());
```

Java – Reading input from file

Files are useful in testing programs because they save the programmer from re-entering (typing) data into the program every time the program is run.

```
import java.util.Scanner;


Scanner in = new Scanner (new File(“filename.txt”));


while(in.hasNext()) { // Iterates each line in the file
    String line = in.nextLine();
}
in.close(); // Don’t forget to close resource leaks
```

Java Exception – best practice for beginner

- Quite difficult for beginner to understand how Java exceptions should be thrown or handled
- E.g. *FileNotFoundException* – happened when you put invalid file path.
- Use *throws FileNotFoundException* or *throws IOException* for now

Tutorial 1

- Due tonight before 23:59pm

Quiz

- Go to www.menti.com and use the code **47 89 50**