Software Construction
Basics of Java Part IV

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

ILOs

- Methods
- Difference between methods, functions and procedures
- Call by value Vs. Call by reference
- Method overloading
- Calling static functions
- Using the class name to resolve conflicts

Java Methods

- A method is a collection of statements that are grouped together to perform an operation.
- Terms: function/methods/procedures:
  - Procedure $\Longrightarrow$ does not return a value
  - Function $\Longrightarrow$ collection of statements that are grouped together to perform an operation.
  - Method $\Longrightarrow$ collection of statements that are grouped together to perform an operation which is part of a class.
  - *Functions as the same as method, but methods are in OOP*
  - So, we will use term *method*.
- Two main types of methods in Java:
  - Static methods $\Longrightarrow$ can be invoked without an object
  - Non-static methods $\Longrightarrow$ which cannot be invoked without an object

Defining a method

```
public static int getMax(int [] data, int start, int end) {
  int i, max;
  for(i = start+1, max = data[start]; i < end; i++)
     if(max < data[i]) max = data[i];
  return max;
}
```

see Functions.java

- **public**: access modifier. Public means the method can be called from any other method (in this class or any other class). Other options:
  - private $\Longrightarrow$ method can only be called from methods in that class
  - protection: Later :)
- **static**: context specifier. Static means the method can be called without an object.
- **int**: return type. Once the method finishes the type of the value it returns.
- **(int [] data, int start, int end)**: list of arguments. Can have any number of arguments separated with commas.

Calling the function

- A *static* method can be invoked without an object
- A *static* method can access arguments passed, static variables and any local variable defined within the function.
- A *static* method can be called with the $< className > . < functionName >$
  - this is done to resolve *name space* collisions
  - a *public*, *static* variable can also be accessed using $< className > . < variableName >$

see Functions.java TestFunc.java

```
int max = getMax(data, 0, data.length);
System.out.println("Max in array = " + max);

int maxVal = Function.getMax(data, 0, data.length);
// static method provided by the Functions class
```

Calling the function: from another class

- this is done to resolve *name space* collisions
- a *public*, *static* variable can also be accessed using $< className > . < variableName >$

see Functions.java TestFunc.java

```
int [] data = {1, 2, -2, 3, 423, 5, -2};
int max = Function.getMax(data, 0, data.length);
System.out.println("Max value in Functions " + Functions.MAX);
// Max variable provided by Functions class
```

- try compiling `TestFunc.java`. Note that

Calling the function: from another class

- this is done to resolve *name space* collisions
- a *public*, *static* variable can also be accessed using $< className > . < variableName >$

see Functions.java TestFunc.java

```
int [] data = {1, 2, -2, 3, 423, 5, -2};
int max = Function.getMax(data, 0, data.length);
System.out.println("Max value in Functions " + Functions.MAX);
// Max variable provided by Functions class
```

- try compiling `TestFunc.java`. Note that

Calling the function: from another class

- this is done to resolve *name space* collisions
- a *public*, *static* variable can also be accessed using $< className > . < variableName >$

see Functions.java TestFunc.java

```
int [] data = {1, 2, -2, 3, 423, 5, -2};
int max = Function.getMax(data, 0, data.length);
System.out.println("Max value in Functions " + Functions.MAX);
// Max variable provided by Functions class
```

- try compiling `TestFunc.java`. Note that

Calling the function: from another class

- this is done to resolve *name space* collisions
- a *public*, *static* variable can also be accessed using $< className > . < variableName >$

see Functions.java TestFunc.java

```
int [] data = {1, 2, -2, 3, 423, 5, -2};
int max = Function.getMax(data, 0, data.length);
System.out.println("Max value in Functions " + Functions.MAX);
// Max variable provided by Functions class
```

- try compiling `TestFunc.java`. Note that

Call by value vs. call by reference

see Calling.java

```
public static void swap(int [] data, int i, int j) {
  int tmp = data[i]; data[i] = data[j]; data[j] = tmp;
}

public static void main(String [] args) {
  int [] data = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  show(data);
  // Swap locations 0 and 1
  swap(data, 0, 1); // data is sent by reference
  show(data);

...
}
```

- Arrays are *passed by reference*
- Objects (anything created by using *new* keyword) are *passed by reference*
- [illegible]

Function overloading

see Calling.java

Basic idea:

- two (or more) methods with the same name but different arguments list and/or return type (parameters)
- correct method will be selected based on the parameters

```
public static void swap(int [] data, int i, int j) {
  int tmp = data[i]; data[i] = data[j]; data[j] = tmp;
}

public static void swap(int a, int b) {
  int tmp = a; a = b; b = tmp;
}

public static void swap(Integer a, Integer b) {
  Integer tmp = a; a = b; b = tmp;
}
```

Function overloading ...

see Calling.java

Basic idea:

- two (or more) methods with the same name but different number/type of arguments
- correct method will be selected based on the parameters
- (cannot overload based on the return type)
- (overloaded methods can have different return types)

```
int [] data = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
show(data);
// Swap locations 0 and 1
swap(data, 0, 1); // data is sent by reference
show(data);

// Swap two variables
int a = 1, b = 0;
System.out.printf("a = %d, b = %d\n", a, b);
swap(a, b); // call by value
System.out.printf("a = %d, b = %d\n", a, b);
```

Overloading continues

see Overloading.java

```java
public static int foo() {
  System.out.println("In int return"); return 1;
}
public static int foo(int i) {
  System.out.println("In int argument, int return"); return i;
}
public static double foo(double d) {
  System.out.println("In double argument, double return");
      return d;
}

public static void main(String [] args) {
  int a = foo();
  int b = foo(a);
  double d = foo(1d);
}
```

ILOs: Revisited

- Methods
- Difference between methods, functions and procedures
- Call by value Vs. Call by reference
- Method overloading
- Calling static functions
- Using the class name to resolve conflicts