Object Oriented Programming (OOP)

Lecture3: Methods and Strings

Prepared by:
Mohamed Mabrouk

Those slides are based on slides by:
Dr. Sherin Mousa and Dr. Sally Saad

Lecture Outline

- Methods
  - Method types
  - Static/final/abstract methods
  - Overloading
  - Parameter passing
  - Constructors
- Strings
  - Conversion between strings and numbers
  - String equality

Methods

Method Types

- Instance methods
- Class methods
- Constructors
- Main method

Method Declaration

- Method declaration has two parts:
  - Method header
  - Method body
- Method header defines method's name, parameters, return type, etc.
- Method body is the actual body of the method that defines its logic

Method Header

- Method header has the following structure:

```
<Modifier> ReturnType MethodName(ParamList)
```

- Types of modifiers are:
  - Access modifier can be `private`, `protected`, or `public`. It is optional and if not provided → `package local`
  - `static` → for the whole class and not for a specific object.
    Can only access static fields and methods of the class

Method Header

- Types of modifiers are:
  - `final` → cannot be overridden in child classes (prevents inheritance)
  - `abstract` → MUST BE OVERRIDEN.
    Has an empty body in parent class but not in child classes

Access Modifiers

- Access modifiers are:
  - `private` → Visible only within the same class and invisible outside it
  - `None` → default visibility. Visible within the same class and all classes within the same package
  - `protected` → Visible within the class and all its subclasses, i.e. inherited classes
  - `public` → Visible anywhere. Be careful when declaring a member as public as it can be modified from outside

Access Modifiers

- protected int add(int num1, int num2){}
  - `protected` → access modifier
  - `int` → return type
  - `add` → method name
  - `int num1, int num2` → parameter list

Static Members

- `static` means that it belongs to the class and not to a specific instance
- `static` methods can access its parameters, local variables and also other static members
- Can be called via class name not object name, e.g. `Student.display()`
- An object can also access static members
- `main` MUST be declared static

Static Member Example

- Implement an instance counter → count the number of objects created of a specific class

Method Call

- To call a method use the dot (.) using either class or object to which the method belongs

```
reference.method(arguments);
```

- Static methods:
  - From outside use class name or object reference to call the method
  - From inside the class the reference has to be omitted
- Non-static methods
  - From outside use object reference to call the method
  - From inside the class the reference has to be omitted

Method Call Example

```
class TestClass{
        public static void method1(){
        }

        public static void method2(){
        }
    }
```

Method Call Example

```
class TestClass{
        public static void method1(){
        }

        public static void method2(){
        }
    }
    ----------------------------------------
    TestClass.methdo1();

    TestClass obj = new TestClass();
    Obj.method2();
```

Method Overloading

- More than one method with same name but with different parameter types or different number of parameters
- Return type *CANNOT* be used to overload a method
- For instance:

```
int add(int num1, int num2){}
int add(int num1, int num2, int num3){}
int add(float num1, float num2){}
```

Method Parameters

- Parameters are placeholders for values that the method should work on
- Can be primitive types, e.g. `int` and `float`, or they can be objects of other classes, e.g. Student, or arrays
- Can have the same name as class fields
- Always passed by value (no passing by reference in Java)

Method Parameters

- If the parameter is an object → it is the object reference is passed (*TRICKY*)
- When the method returns, the passed-in reference still references the same object
- The *object's fields* *can* be changed in the method

Parameter Passing example

```
public static void main(String[] args){
      Student stud = new Student();
       stud.setName("A");

      System.out.println(stud.getName());
      changeName(stud);
      System.out.println(stud.getName());
}

public static void changeName(Student student){
      student.setName("B");
}
```

Parameter Passing example

```
public static void main(String[] args){
      Student stud = new Student();
       stud.setName("A");

      System.out.println(stud.getName());
      changeName(stud);
      System.out.println(stud.getName());
}

public static void changeName(Student student){
      student  = new Student();
      student.setName("B");
}
```

Method Parameters

- The object fields can be changed in methods → will impact the passed object
- The object itself cannot refer to a new memory address → After method call it will retain its original address

`this` Keyword

- Refers to the object from which it is called
- Can be used in case of members are *SHADOWED* by method/constructor parameters

```
public class Student {
   String name;
   float marks;

  public Student(String name, float marks){
     this.name = name;
     this.marks = marks;
  }
}
```

`this` Keyword

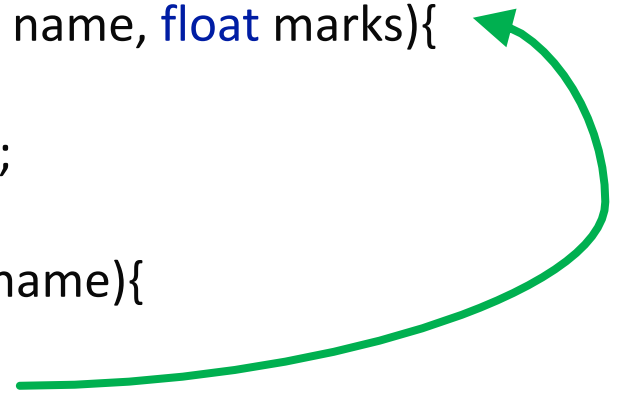
- Can also be used to call another constructor

```
public class Student {
    String name;
    float marks;

   public Student(String name, float marks){
       this.name = name;
       this.marks = marks;
   }
  public Student(String name){
       this(name, 0.0F);
   }
}
```

`this` Keyword

- Can also be used to call another constructor

```
public class Student {
    String name;
    float marks;

   public Student(String name, float marks){
       this.name = name;
       this.marks = marks;
   }
  public Student(String name){
       this(name, 0.0F);
   }
}
```



`main` Method

- Main method should be:
  `public static void main(String[] args)`
- Why `public`?
- Why `static`?
- Why `String[] args`?

`main` Method

- Main method should be:
  `public static void main(String[] args)`
- Why `public`? → Can be accessed from outside
- Why `static`?
- Why `String[] args`?

`main` Method

- Main method should be:
  `public static void main(String[] args)`
- Why `public`? → Can be accessed from outside
- Why `static`? → No instantiation required
- Why `String[] args`?

`main` Method

- Main method should be:
  `public static void main(String[] args)`
- Why `public`? → Can be accessed from outside
- Why `static`? → No instantiation required
- Why `String[] args`? → Pass parameters to the program

`main` Method

- Can be used to pass parameters to the program

```
Public class Greetings {
public static void main(String[] args){
      String name = args[0];
       System.out.println("Hello " + name);
   }
}
```

`main` Method

- Can be used to pass parameters to the program

```
Public class Greetings {
public static void main(String[] args){
        String name = args[0];
        System.out.println("Hello " + name);
    }
}
```

- java Greetings Mohamed

`main` Method

- All parameters should be strings
- Can be converted to integers or floats, etc.

Destroying Objects

- When an object is eligible for garbage collection → it is deleted by garbage collector
- The object is eligible for garbage collection in case:
  - A reference to it is set to NULL
  - The reference to the object is made to refer to another object
- How to force garbage collector to work → `System.gc`
- *DO NOT CALL IT YOURSELF*

Strings

Strings

- Is a sequence of characters
  `String str = "Hello";`
- Is it a primitive type or a reference type?
- Strings are immutable

Strings

- Is a sequence of characters
  `String str = "Hello";`
- Is it a primitive type or a reference type?
- Strings are immutable → what does that mean?

Strings

- Is a sequence of characters
  `String str = "Hello";`
- Is it a primitive type or a reference type?
- Strings are immutable → what does that mean?
- Immutable means that its value cannot be changed

Strings

- Is a sequence of characters
  `String str = "Hello";`
- Is it a primitive type or a reference type?
- Strings are immutable → what does that mean?
- Immutable means that its value cannot be changed → What happens if its value has changed?

Strings

```
public static void main(String[] args){
      String str = "Hello";
      char[] chars = { 'H', 'i' } ;

      String s1 = new String();
   }
```

Strings

```
public static void main(String[] args){
      String str = "Hello";
      char[] chars = { 'H', 'i' } ;

      String s1 = new String();
      String s2 = new String(str);
   }
```

Strings

```
public static void main(String[] args){
      String str = "Hello";
      char[] chars = { 'H', 'i' } ;

      String s1 = new String();
      String s2 = new String(str);
      String s3 = new String(chars);
   }
```

Conversion to String

- To convert any object to string you should override method `toString()`
- Many standard Java classes already override `toString()`
- That enables printing any object to the screen as a string

String Comparison

- Can ‘==‘ be used for comparing strings?

String Comparison

- '==' Operator *CANNOT* be used to compare strings
- It compares object references, i.e. their addresses
- To compare string contents `equals()` method can be used
- <, <=, >, and >= cannot be used to compare strings

String Comparison

- To compare strings `compareTo()` methods can be used
- It returns -1, 0, or 1
  - -1 if s1 < s2
  - 0 if s1 = s2
  - 1 if s1 > s2

String Comparison

```
public static void main(String[] args){
      String s1 = “Hello”;
      String s2 = “Hi”;
      int val = s1.compareTo(s2);
      System.out.println(val);
   }
```

Useful String Methods

- String class has many methods. To name a few:
  - `substring`: returns substring starting at position and ends at anther position
  - `indexOf`: finds a substring if exists and returns its position, and -1 if not found
  - `lastIndexOf`: finds a substring if exists and returns its position but from end, and -1 if not found
  - `replace`: replaces a substring in the string with another substring
  - `split`: splits a string with a specific separator(s)
  - `startsWith`: checks if a string starts with the given substring
  - `endsWith`: checks if a string ends with the given substring

Thank You!