Structured Programming

Lecture 2

Dr. Obead Alhadreti

Outline

- Modifiers
- Methods

Modifiers

Modifiers

- Modifiers are keywords that are added to change meaning of a definition. In Java, modifiers are catagorized into two types,

1. Access modifiers
2. Non-access modifiers

1. Access modifiers

- The access to **classes**, **variables and methods** are regulated using access modifiers.

- i.e. a class can control what information or data can be accessible by other classes.

- To take advantage of *encapsulation*, you should minimize access whenever possible.

1. Access modifiers

1. **public:** are **visible to any class in the Java program**, whether these classes are in the same package or in another package.

2. **private:** they **cannot** be accesses by **anywhere outside** the enclosing class.

* A class **can not** be declared as private

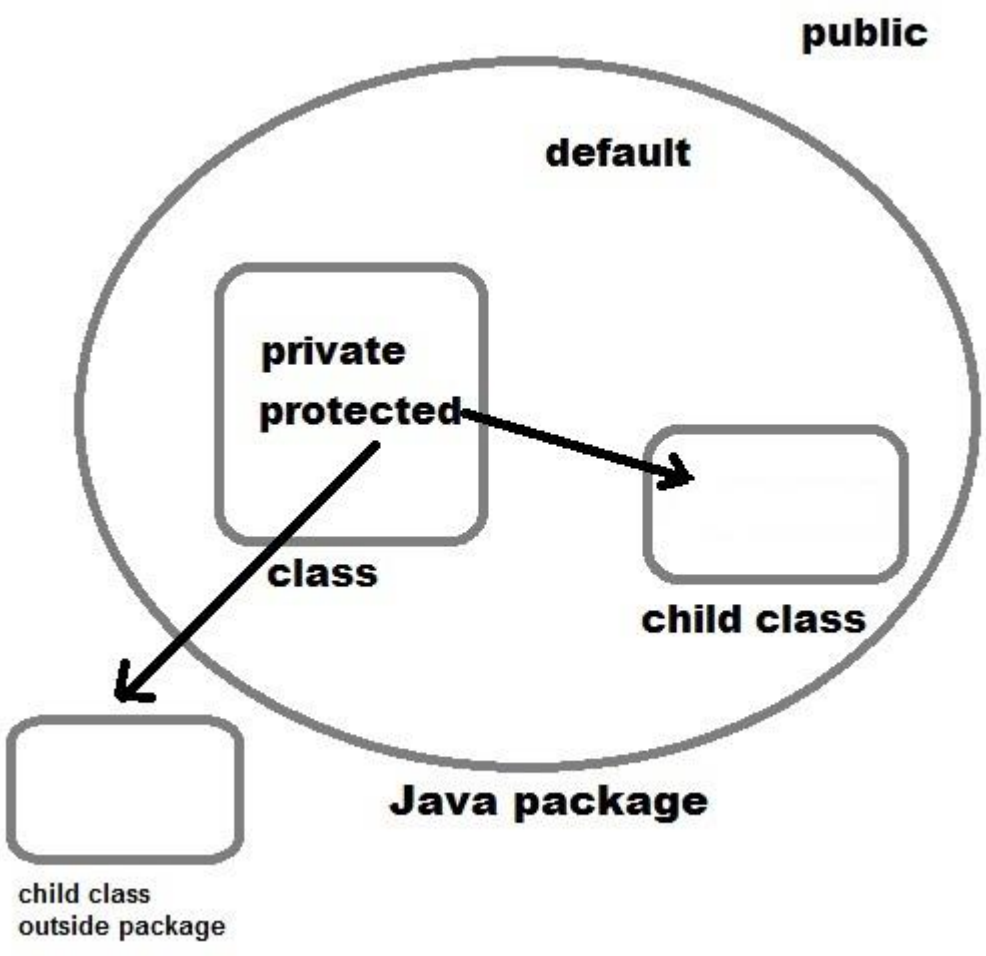
* A standard design strategy is to make all variables private, and methods public.

1. Access modifiers

3. **protected:Variables and methods** declared protected in a super class can be **accessed only by subclasses** in other packages. **Classes in the same package** can also access protected **variables**, and **methods** as well, even if they are not a subclass of the protected member's class.

4. **default:** when no access modifier is present, any class, variable, and methodthat has no declared access modifier is accessible only by **classes in the same package (known as** package private**)**.

1. Access modifiers



1. Access modifiers

	Can be	Better to be
Class	Public or default	Public or default
Variables	Public, default, private, or protected	Private
Methods	Public, default, private, or protected	Public

2. Non-access modifiers

- Non-access modifiers do not change the accessibility of variables and methods, but they do provide them special properties. There are two main types of non-access modifiers:

1. Static
2. Final

2. Non access modifiers

1. Static Modifier

Static modifier is used to create class variable and class methods which can be accessed without instance of a class.

A. Static Variables: Static variables (also known as class variables or shared variables), which can be *access variables directly using class name*, have only one single storage. All the object of the class having static variable will have the same instance of static variable. Static variables represent common property of a class. It saves memory. Static variables are owned by class rather than by its individual instances (objects). However, *static variables can change values*.

2. Non access modifiers

- Accessing a static variable
  - **ClassName.myStaticVariable**

- Suppose there are 100 employee in a company. All employee have its unique name and employee id but company name will be same all 100 employee. Here company name is the common property. So if you create a class to store employee detail, companyName field will be mark as static.

Example of static variables

Start Page | EmployeeRecords.java | **Employee.java**

```
1   package employeerecords;
2
3   public class Employee {
4   int id;
5   String name;
6   static String companyName = "Apple";
7
8       public void show()
9       {
10          System.out.println (id+" "+name+" "+companyName);
11      }
12  }
13
```

Example of static variables

Start Page | EmployeeRecords.java | Employee.java

```
1   package employeerecords;
2
3   public class EmployeeRecords {
4
5   public static void main(String[] args) {
6
7           Employee e1 = new Employee();
8           e1.id = 104;
9           e1.name = "Ahmed";
10
11
12          Employee e2 = new Employee();
13          e2.id = 108;
14          e2.name = "Fatma";
15
16          e1.show();
17          e2.show();
18
19
20      }
21
22  }
```

Output - EmployeeRecords (run)

```
run:
104 Ahmed Apple
108 Fatma Apple
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example of static variables

Start Page | EmployeeRecords.java | Employee.java

```
1   package employeerecords;
2
3   public class EmployeeRecords {
4
5   public static void main(String[] args) {
6
7           Employee.companyName= "Samsung";
8
9           Employee e1 = new Employee();
10          e1.id = 104;
11          e1.name = "Ahmed";
12
13
14          Employee e2 = new Employee();
15          e2.id = 108;
16          e2.name = "Fatma";
17
18          e1.show();
19          e2.show();
20
21
22      }
23
24  }
25
```

Output - EmployeeRecords (run)

```
run:
104 Ahmed Samsung
108 Fatma Samsung
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Non access modifiers

Static variable vs Instance Variable

Static variable	Instance Variable
Represent common property	Represent unique property
Accessed using class name	Accessed using object
get memory only once	get new memory each time a new object is created

2. Non access modifiers

B. Static Method:

- Sometimes a method performs a task that does not depend on an object. In other words, it applies to the class in which it's declared as a whole.

- To declare a method as static, place the keyword static before the return type in the method's declaration.

- Calling a static method
  - **ClassName.myMethodName()**

2. Non access modifiers

- Static methods use no instance variables. They work with static variables, and can also take input from the parameters, perform actions on it, then return some result.

- **main**() method is the most common example of static method.

- Static methods do not need instance of its class for being accessed.

Example

Start Page | Main.java | Person.java

```
1   package personrecords;
2
3   public class Person {
4
5     String name;
6     int age;
7     public static String  companyName= "Apple";
8
9     void show () {
10    System.out.println("Name: " +name);
11    System.out.println("Age: " +age);
12    System.out.println();
13    }
14
15      public static void printInfo() {
16          System.out.println("Company Name: " +companyName);
17          System.out.println();
18      }
19  }
```

Example

Start Page | Main.java | Person.java

```
1   package personrecords;
2
3   public class Main {
4
5   public static void main(String[] args) {
6
7   Person a1 = new Person ();
8   Person a2 = new Person ();
9
10  a1.name= "Ahmed";
11  a1.age = 33;
12
13  a2.name = "Nourah";
14  a2.age = 21;
15
16  a1.show ();
17  a2.show ();
18
19  Person.printInfo();
20
21      }
22
23  }
```

Output - PersonRecords (run)

```
run:
Name: Ahmed
Age: 33

Name: Nourah
Age: 21

Company Name: Apple

BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Non access modifiers

2. Final Modifier

Final modifier is used to declare a field as final i.e. it prevents its content from being modified. Final field must be initialized when it is declared. Final keyword can be used with a variable, a method or a class.

A. Final Variable: When a variable is declared as final, then its value cannot be changed. The variable acts like a constant. It also cannot be used without creating instance of a class.

```
final int a = 5;
```

2. Non access modifiers

B. Final Method: When a method is declared as final, then that method cannot be overridden (redefined). A final method can be inherited/used in the subclass, but it cannot be overridden (redefine).

C. Final Class: A class can also be declared as final. A class declared as final cannot be inherited.

2. Non access modifiers

- What cannot be changed and will be *accessible* without creating instance of a class will be a *static final variable* .

- The names of variables declared class constants should be all uppercase with words separated by underscores ("_").

```
static final int MIN_WIDTH = 4;

static final int MAX_WIDTH = 999;

static final int GET_THE_CPU = 1;
```

Methods

Methods

- A **method** is a set of code which is referred to by name and can be called at any point in a program simply by utilizing the method's name to perform certain task. Think of a method as a subprogram that acts on data and often returns a value.

- Every class consists of one or more methods .A java applications must contain one ***main*** method in a public class. Execution always begins with ***main*** method.

- Methods are time savers, in that they allow for the repetition of sections of code without retyping the code.

Methods

- There are two basic types of methods:

1. Built-in: Build-in methods are part of the compiler package, such as System.out.println( ) and System.exit(0).

2. User-defined: User-defined methods are created by you, the programmer. These methods take-on names that you assign to them and perform tasks that you create.

Method Declaration

```
modifier returnType methodName(Parameters List){

      // Method Body

}
```

- Methods typically starts by the keywords public static (modifier). The only required elements of a method declaration are the method's return type, **name**, a pair of parentheses, (), and a body between braces, {}.

- The return type specifies the type of data the method returns after performing its task.

Method Declaration

- Return type void indicates that a method will perform a task but will not return (i.e., give back) any information to its calling method when it completes its task.

- Method **name** follows the return type. By convention, method names begin with a lowercase first letter and subsequent words in the name begin with a capital letter (known as camelCase).

- Empty parentheses after the method name indicate that the method does not require additional information to perform its task.

Example

Start Page | StudentRecords.java | Student.java

```
1   package studentrecords;
2
3   public class Student {
4   String name;
5   String sex;
6   int age;
7
8   void printInfo () {
9
10  System.out.println ("Name:" +name);
11  System.out.println ("Sex:" +sex);
12  System.out.println ("Age:" +age);
13  System.out.println ();
14
15  }
16  }
```

Example

Start Page | StudentRecords.java | Student.java

```
1   package studentrecords;
2
3   public class StudentRecords {
4
5       public static void main(String[] args) {
6       Student s1 = new Student ();
7       Student s2 = new Student ();
8
9       s1.name = "Nourah";
10      s1.sex = "Female";
11      s1.age = 19;
12
13      s2.name = "Ahmed";
14      s2.sex = "Male";
15      s2.age = 21;
16
17      s1.printInfo ();
18      s2.printInfo ();
19      }
20
21  }
```

Output - StudentRecords (run)

```
run:
Name:Nourah
Sex:Female
Age:19

Name:Ahmed
Sex:Male
Age:21

BUILD SUCCESSFUL (total time: 0 seconds)
```