

CSEN 202 – Introduction to Computer Programming

Lecture 7: Classes and objects I

Prof. Dr. Slim Abdennadher and
Dr Mohammed Abdel Megeed Salem
`slim.abdennadher@guc.edu.eg`

German University Cairo, Faculty of Media Engineering and Technology

April 21 - April 26, 2018

A world of objects

- Forget programming for a while.

¹Read: Descartes. “*Discours de la méthode*”, 1637

A world of objects

- Forget programming for a while.
- Think about things in the world that are **objects**, and things that are not objects.

¹Read: Descartes. “*Discours de la méthode*”, 1637

A world of objects

- Forget programming for a while.
- Think about things in the world that are **objects**, and things that are not objects.
- It is easier to list things that are objects than to list things that are not objects.

¹Read: Descartes. “*Discours de la méthode*”, 1637

A world of objects

- Forget programming for a while.
- Think about things in the world that are **objects**, and things that are not objects.
- It is easier to list things that are objects than to list things that are not objects.
- **Descartes**: Humans view the world in object oriented terms: human brain wants to think about objects, and our thoughts and memories are organized into objects and their relationships.¹

¹Read: Descartes. “*Discours de la méthode*”, 1637

A world of objects

- Forget programming for a while.
- Think about things in the world that are **objects**, and things that are not objects.
- It is easier to list things that are objects than to list things that are not objects.
- **Descartes**: Humans view the world in object oriented terms: human brain wants to think about objects, and our thoughts and memories are organized into objects and their relationships.¹
- One idea of **object-oriented software** is to organize software in a way that matches the thinking style of our object-oriented brains.

¹Read: Descartes. "*Discours de la méthode*", 1637

A world of objects

Example

- **Student** can be described by name, gender, application number, ...
- **Car** can be described by model, make, year, ...

Classes and objects

- A **class**: a **template** for **creating objects** with similar features. It contains variables to represent the attributes and methods to represent the behavior of the objects.

Classes and objects

- A **class**: a **template** for **creating objects** with similar features. It contains variables to represent the attributes and methods to represent the behavior of the objects.
- An **object**: **entity** that you can manipulate in your programs (by invoking methods)

Classes and objects

- A **class**: a **template** for **creating objects** with similar features. It contains variables to represent the attributes and methods to represent the behavior of the objects.
- An **object**: **entity** that you can manipulate in your programs (by invoking methods)
- When a Java application is being run, objects are **created** and their methods are **invoked** (are run.)

Classes and objects

- A **class**: a **template** for **creating objects** with similar features. It contains variables to represent the attributes and methods to represent the behavior of the objects.
- An **object**: **entity** that you can manipulate in your programs (by invoking methods)
- When a Java application is being run, objects are **created** and their methods are **invoked** (are run.)
- A programmer may **define a class** using Java, or may **use predefined classes** that come in class libraries.

Classes and objects

- A **class**: a **template** for **creating objects** with similar features. It contains variables to represent the attributes and methods to represent the behavior of the objects.
- An **object**: **entity** that you can manipulate in your programs (by invoking methods)
- When a Java application is being run, objects are **created** and their methods are **invoked** (are run.)
- A programmer may **define a class** using Java, or may **use predefined classes** that come in class libraries.
- **Creating an object** is called **instantiation**.

Classes and objects

- Think of the **class** as a **mold** used to create instances



Classes and objects

- Think of the **class** as a **mold** used to create instances



- Think of the **object** as one **cast** created from the mold



Classes and objects

- Think of the **class** as a **mold** used to create instances



- Think of the **object** as one **cast** created from the mold



- Naturally, **multiple** objects can be created from one class: 🍪, 🍪, 🍪, 🍪, 🍪, 🍪, 🍪, 🍪, 🍪, ...

Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.



Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.

- **Static** is everything that belongs to the **class** (to the mold).



Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.

- **Static** is everything that belongs to the **class** (to the mold).
- **Not static** (*i. e.*, **dynamic**) is everything that belongs to individual **objects**.



Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.

- **Static** is everything that belongs to the **class** (to the mold).
- **static methods / variables** can be called/ manipulated through the class.
- **Not static** (*i. e.*, **dynamic**) is everything that belongs to individual **objects**.



Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.

- **Static** is everything that belongs to the **class** (to the mold).
- **static methods / variables** can be called/ manipulated through the class.
- **Not static** (*i. e.*, **dynamic**) is everything that belongs to individual **objects**.
- **instance methods / variables** exist only through an object.



Classes and objects

Recall: So far we used the keyword “**static**” on **everything**.

- **Static** is everything that belongs to the **class** (to the mold).
- **static methods / variables** can be called/ manipulated through the class.
- **Not static** (*i. e.*, **dynamic**) is everything that belongs to individual **objects**.
- **instance methods / variables** exist only through an object.



Why does the **main**-method have to be defined as **static**?

Classes and objects—examples

An **object** of class **Employee** has

Classes and objects—examples

An **object** of class `Employee` has

- **Attributes** (which are like adjectives)
 - `age`
 - `educationalDegrees`
 - `yearsOfExperience`
 - `jobTitle`
 - `emailAddress`

Classes and objects—examples

An **object** of class `Employee` has

- **Attributes** (which are like adjectives)

- `age`
- `educationalDegrees`
- `yearsOfExperience`
- `jobTitle`
- `emailAddress`

- **Methods** (or actions) the object can perform or undertake while on the job:

- `wearCompanyT-shirt()`
- `emailJokesToFriends()`
- `SurfInternet()`
- `eatJunkfood()`

Classes and objects—examples

An **object** of class **Car** has

Classes and objects—examples

An **object** of class `Car` has

- **Attributes**

- `year`
- `make`
- `model`
- `topSpeed`
- `isRunning`

Classes and objects—examples

An **object** of class `Car` has

- **Attributes**

- `year`
- `make`
- `model`
- `topSpeed`
- `isRunning`

- **Methods** (which correspond to actions the driver might take):

- `start()`
- `stop()`
- `isRunning()`
- `turnLeft()`

Classes and objects—examples

■ **Attributes:**

- `make`: of type `String`
- `model`: of type `String`
- `year`: of type `int`
- `isRunning`: of type `boolean`

Classes and objects—examples

■ Attributes:

- `make`: of type `String`
- `model`: of type `String`
- `year`: of type `int`
- `isRunning`: of type `boolean`

■ Methods:

- `start()`: the `start` method starts the car by setting its boolean attribute to `true`; the method does not return anything.
- `stop()`: the `stop` method stops the car by setting its boolean attribute to `false`; this method does not return anything.
- `isRunning()`: the `isRunning` method tells you whether or not the car is running, by returning a boolean value (`true` if it the car is running).

Instance variables

Instance variables are variables to store the state (attributes) of an object.

```
access specifier class class name {  
    ...  
    access specifier variable type variable name;  
    ...  
}
```

Instance variables

Instance variables are variables to store the state (attributes) of an object.

```
access specifier class class name {  
    ...  
    access specifier variable type variable name;  
    ...  
}
```

- An access specifier (usually “**private**”)

Instance variables

Instance variables are variables to store the state (attributes) of an object.

```
access specifier class class name {  
    ...  
    access specifier variable type variable name;  
    ...  
}
```

- An **access specifier** (usually “**private**”)
- The **type** of the variable

Instance variables

Instance variables are variables to store the state (attributes) of an object.

```
access specifier class class name {  
    ...  
    access specifier variable type variable name;  
    ...  
}
```

- An **access specifier** (usually “**private**”)
- The **type** of the variable
- The **name** of the variable

Instance variables—example

```
public class Car {  
    private String make;  
    private String model;  
    private int year;  
    private boolean isRunning = false;  
    ...  
}
```

How to construct an object?

The **constructor** is a **special type of method**

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name!**

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name!**
- Returns an **instance** of the class (an **object**)

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name**!
- **Returns** an **instance** of the class (an **object**)
- **Initializes** the instance variables (set certain values for the instance at creation-time)

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name!**
- **Returns** an **instance** of the class (an **object**)
- **Initializes** the instance variables (set certain values for the instance at creation-time)
- Can take any number of parameters

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name**!
- **Returns** an **instance** of the class (an **object**)
- **Initializes** the instance variables (set certain values for the instance at creation-time)
- Can take any number of parameters
- Can take any type of parameters

How to construct an object?

The **constructor** is a **special type of method**

- Does **not** have a **name**!
- Returns an **instance** of the class (an **object**)
- **Initializes** the instance variables (set certain values for the instance at creation-time)
- Can take any number of parameters
- Can take any type of parameters

```
access specifier class class name {  
    access specifier class name (parameter type parameter name, ...) {  
        ...  
    }  
}
```

How to construct an object?

```
public class Car {  
    private String make;  
    private String model;  
    private int year;  
    private boolean isRunning = false;  
  
    public Car (String mke, String mdl, int y) {  
        make = mke;  
        model = mdl;  
        year = y;  
    }  
  
    ...  
}
```

Some methods

```
public boolean isRunning () {  
    return isRunning;  
}
```

```
public void start () {  
    if (isRunning == false) {  
        System.out.println ("Starting_the_car.");  
        isRunning = true;  
    }  
}
```

```
public void stop () {  
    if (isRunning == true) {  
        System.out.println ("Stopping_the_car.");  
        isRunning = false;  
    }  
}
```

Instantiating a class

To create an **instance** (or object) from the class, we use the keyword **new** followed by a call to the constructor.

Instantiating a class

To create an **instance** (or object) from the class, we use the keyword **new** followed by a call to the constructor.

- Syntax: `class name variable name = new constructor;`

Instantiating a class

To create an **instance** (or object) from the class, we use the keyword **new** followed by a call to the constructor.

- Syntax: `class name variable name = new constructor;`
- Result: The constructor constructs the object and returns a **reference** to that newly created object.

Instantiating a class

To create an **instance** (or object) from the class, we use the keyword **new** followed by a call to the constructor.

- Syntax: `class name variable name = new constructor;`
- Result: The constructor constructs the object and returns a **reference** to that newly created object.
- Example:

```
Car c = new Car ("VW", "Golf", 1992);
```

Instantiating a class

```
public class Tester {  
  
    public static void main(String[] args) {  
        Car c = new Car ("VW", "Golf", 1992);  
  
        System.out.println (c.isRunning());  
        c.start ();  
        System.out.println (c.isRunning());  
        c.start ();  
        c.stop ();  
        System.out.println (c.isRunning());  
    }  
  
}
```


Testing a predefined class

```
public class StringTester {  
  
    public static void main(String[] args) {  
        String s; // s is a variable that refers to an object (reference),  
                  // but no object is created yet.  
        int len; // len is a variable of the primitive type int  
  
        s = new String("German_university_in_Cairo"); // create a new  
                                                       // object of type string and  
                                                       // assign its reference to s  
        len = s.length(); // invoke the method length()  
                          // of s  
  
        System.out.println("The_string_is_" + len + "_characters_long");  
    }  
}
```

Person

- All persons are described by a common set of properties or **fields** (Instance variables):
 - Name
 - Year of birth

Person

- All persons are described by a common set of properties or **fields** (Instance variables):
 - Name
 - Year of birth
- The **object type** is based on the names and types of its fields.

Person

- All persons are described by a common set of properties or **fields** (Instance variables):
 - Name
 - Year of birth
- The **object type** is based on the names and types of its fields.
- The main role of **classes** is to define types of objects

```
public class Person {  
    String name;  
    int yearOfBirth;
```

Person

- All persons are described by a common set of properties or **fields** (Instance variables):
 - Name
 - Year of birth
- The **object type** is based on the names and types of its fields.
- The main role of **classes** is to define types of objects

```
public class Person {  
    String name;  
    int yearOfBirth;
```

- **Each instance** of this class (object of this type) will have its **own** copies of the instance variables (field values)

Person

■ A constructor

```
public class Person {  
    String name;  
    int yearOfBirth;  
  
    public Person (String n, int y) {  
        name = n;  
        yearOfBirth = y;  
    }  
}
```

...

Instance methods

Each instance of person will have a copy of these instance methods:

```
public void display () {  
    System.out.println ("Name:_ " + name);  
    System.out.println ("Year_of_birth:_ " + yearOfBirth);  
}
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
public int getYearOfBirth() {  
    return yearOfBirth;  
}
```

Instances

Creating and testing a several instances:

```
public class PersonTester {  
  
    public static void main(String[] args) {  
        Person lect = new Person ("Georg", 1973);  
        lect.display ();  
  
        lect.setName("Slim");  
        lect.setYearOfBirth(1967);  
  
        lect.display();  
  
        Person pres = new Person ("Barak", 1961);  
        pres.display();  
  
        System.out.println (lect + ",_" + pres);  
    }  
}
```


Coming up

- Next topic: Classes and objects II