# Object Oriented Programming (OOP)

## Lecture2: Java Syntax; Control Structures, Classes, and Objects

Prepared by:
Mohamed Mabrouk

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

# Lecture Outline

- Branching: `if`, and `switch`
- Loops: `for, foreach, while,` and `do-while`
- Loop Control: `break`, and `continue`
- Arrays
- Classes and Objects
- Constructors
- Packages
- Encapsulation and access modifiers

# Branching

# Branching: if-else

```
if(condition){
Statement 1
Statement 2
…
Statement n
} else {
Statement 1
Statement 2
…
Statement n
}
```

# Branching: if-else

- There can be many if-else control structures
- Always enclose if or else body in braces

# Branching: if-else

```
public static boolean isNegative(int num){
    if(num < 0){
        return true;
    } else {
        return false;
    }
}
```

# Loops

# Loops: for

```
for (int i = 0; i < 10; i++) {
      Statement 1
      Statement 2
      …
      Statement n
}
```

# Loops: foreach

- is used to traverse an array or a collection in Java
- easier to use than simple for loop
- No loop variable

```
int[] arr = new int[10];

for (int item:arr) {
	System.out.println(item);
}
```

# Loops: while and do-while

```
int i = 0;
while(i<10){
        Statement 1
        Statement 2
        …
        Statement n

        i++;
}
```

# Loops: while and do-while

```
int i = 0;
do {
      Statement 1
      Statement 2
      …
      Statement n

      i++;
} while(i<10);
```

# Loop Control

# Loop Control

- `break` can be used to end a loop
- `continue` is used to jump to the loop start
- Limit the number of `break/continue` statement to 1 per loop

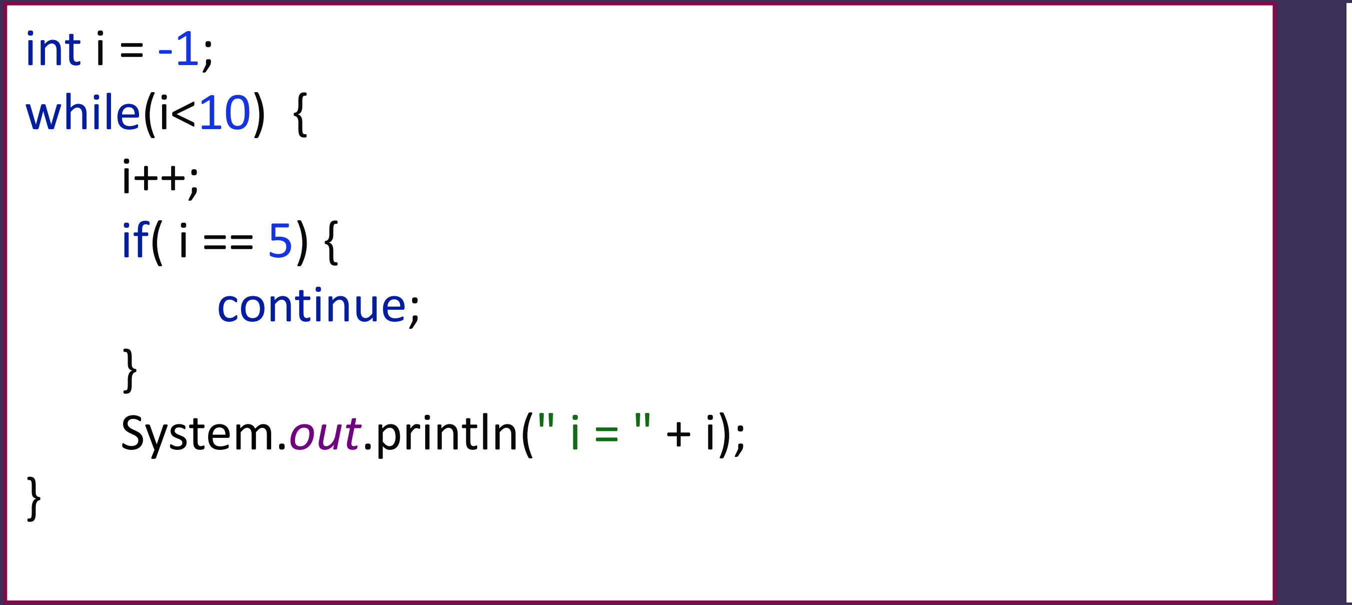
# Loop Control

- What is the output?

```
int i = 0;
while(i<10)  {
        if( i == 5) {
                break;
        }
        System.out.println(" i = " + i);
        i++;
}
```

# Loop Control

- What is the output?



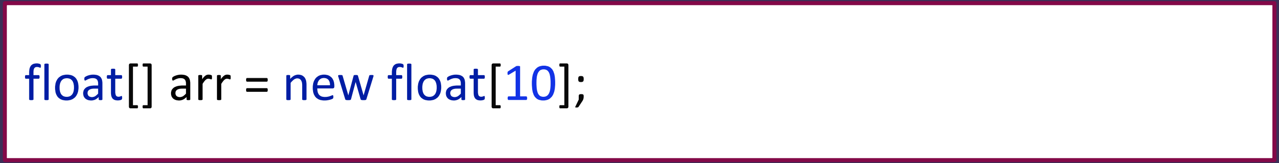
```
int i = -1;
while(i<10)  {
        i++;
        if( i == 5) {
                continue;
        }
        System.out.println(" i = " + i);
}
```

# Arrays

# Arrays

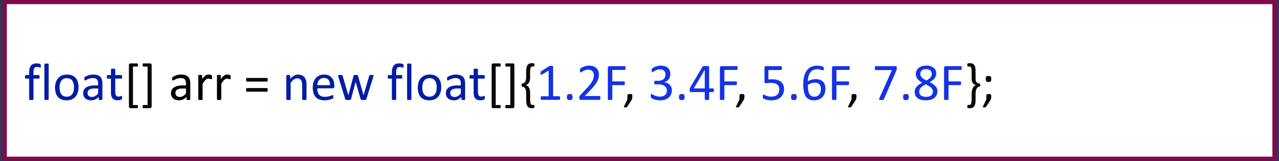
- Is simply a collection of items of the same type
- Has a fixed size
- Can hold any type, including simple types, e.g. int and float, or complex types, e.g. Student or Car
- Only holds references, i.e. does not hold the actual objects
- If any of position is not initialized, it is *NULL*

# Arrays



```
float[] arr = new float[10];
```

- Creates a non-initialized array



```
float[] arr = new float[]{1.2F, 3.4F, 5.6F, 7.8F};
```

- Creates an initialized array with the values specified
- Foreach loops can be also used to iterate overt its members

# Classes & Objects

# Classes & Objects

- A class constitutes the blueprint of a specific type, e.g. Car or Student
- Contains data members (fields) and methods to work on these data members
- Defines various levels of hiding to protect its own fields and methods
- Can be used to create hierarchy, i.e. levels of inheritance among classes
- May also contain inner classes

# Classes & Objects

- An object is an *instance* of s specific class
- It reserves memory in the system
- Can be used do the real job of the class it represents
- Can be instantiated using keyword `new`
- If not instantiated it will be NULL

# Basic Class Syntax

```
modifier class Classname {
    modifier data-type field_1;
    ...
    modifier data-type field_n;

    modifier Constructor_1(parameters) {
    }

    modifier Constructor_n(parameters) {
    }

    modifier Return-Type method_1(parameters) {
        //statements
    }
     ...
    modifier Return-Type method_n(parameters) {
        //statements
    }
```

# Example Class

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

   public Student(String n, float m){
       name = n;
       marks = m;
   }
  public float addMarks(float m){
       marks += m;
       return marks;
   }
}
```

# Example Class

Access Modifiers

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

   public Student(String n, float m){
        name = n;
        marks = m;
    }
  public float addMarks(float m){
        marks += m;
        return marks;
    }
}
```

# Example Class

Class Name

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

   public Student(String n, float m){
       name = n;
       marks = m;
   }
  public float addMarks(float m){
       marks += m;
       return marks;
   }
}
```

# Example Class

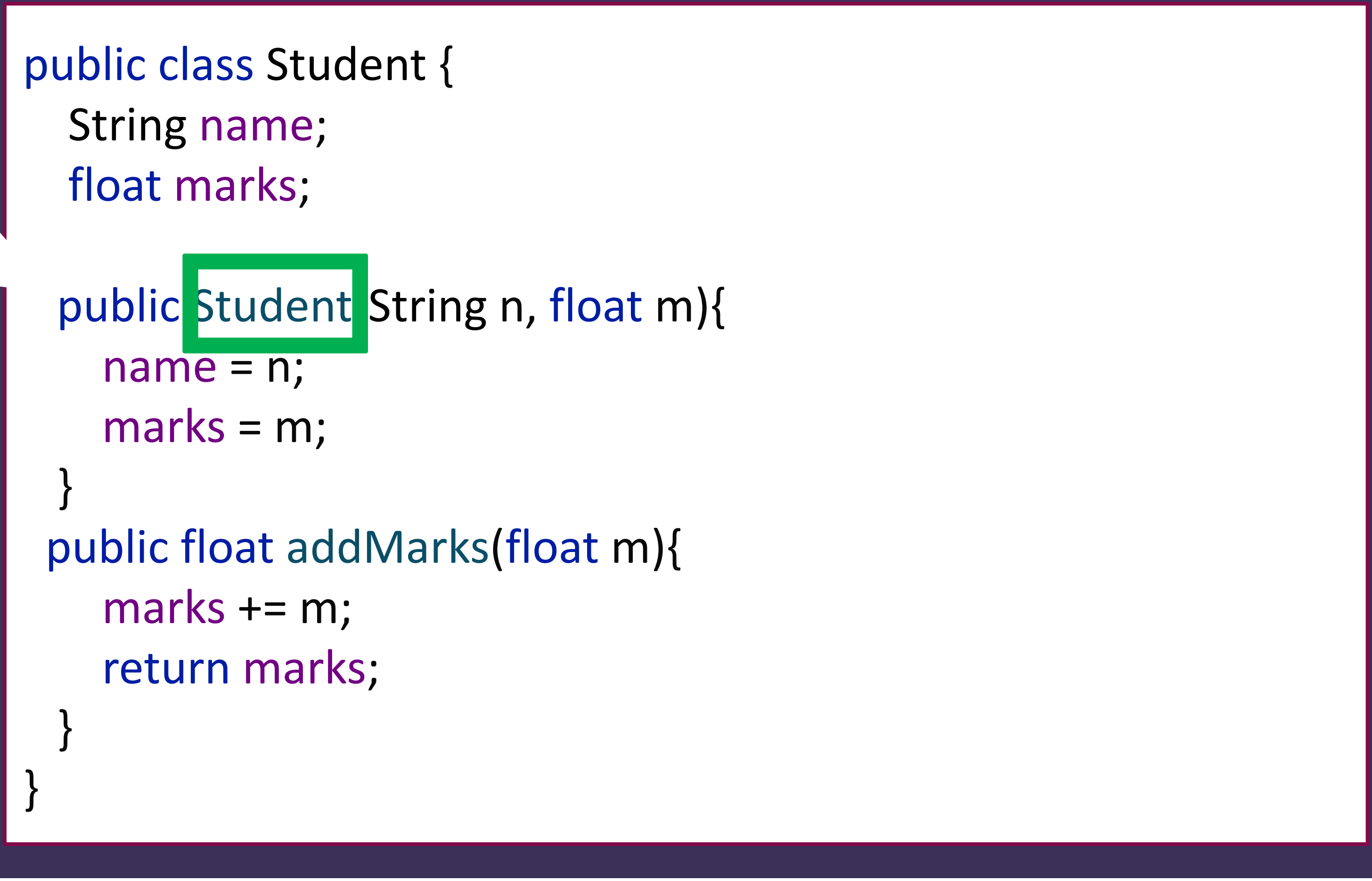
Fields

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

   public Student(String n, float m){
       name = n;
       marks = m;
   }
  public float addMarks(float m){
       marks += m;
       return marks;
   }
}
```

# Example Class

Constructor



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

   public Student(String n, float m){
       name = n;
       marks = m;
   }
  public float addMarks(float m){
       marks += m;
       return marks;
   }
}
```

# Example Class

Methods

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

   public Student(String n, float m){
       name = n;
       marks = m;
   }
  public float addMarks(float m){
       marks += m;
       return marks;
   }
}
```

# Classes & Objects

- `new` keyword can be used to instantiate and object of a class, e.g.

  ```
  Student stud = new Student(“John Smith”,
  75);
  ```

- Dot operator can be used to access fields and methods, e.g.

  ```
  stud.addMarks(10.5);
  ```

# Constructors

- Each class <u>*MUST*</u> have at least one constructor
- Can be many constructors in the same class
- Have no return type
- Must have the exact same name of the class

# Constructors

- If none is defined → compiler creates one with no parameters called *<u>default constructor</u>*
- *<u>default constructor</u>*
    - initializes fields with their default values → zero for numeric types, and false for booleans, and null for object references
    - Calls the constructor of the parent class implicitly
    - To call parent class constructor you can use `super();`

# Unified Modeling Language (UML)

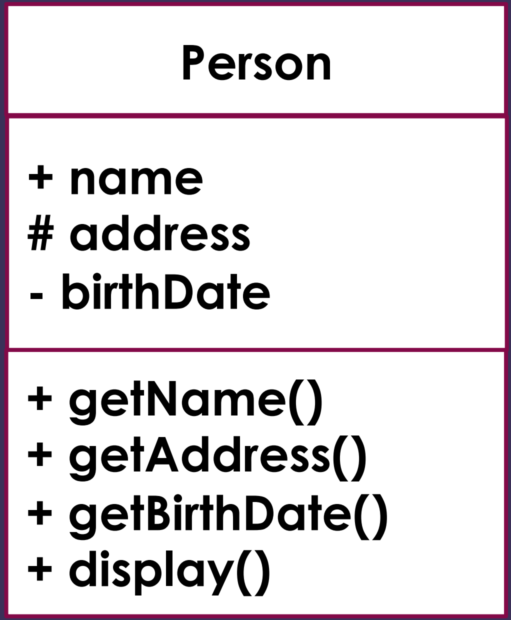
# Unified Modeling Language (UML)

- Object oriented modeling language
- Convenient way of visualizing classes, objects, and relationships among system classes
- Is not bound to a specific language, i.e. not necessary Java
- Helps getting an overview on the system and its inherent structure and hierarchy

# Class Diagram

- Describes the classes of the system and the relationships among them
- Describes attributes (fields), and operations (methods) of the class
- Represented in UML by a rectangle, usually divided into three sections
    1. Class name
    2. Attributes (fields)
    3. Operations (methods)

# Class Diagram



**Person**

**+ name**<br>
**# address**<br>
**- birthDate**

**+ getName()**<br>
**+ getAddress()**<br>
**+ getBirthDate()**<br>
**+ display()**

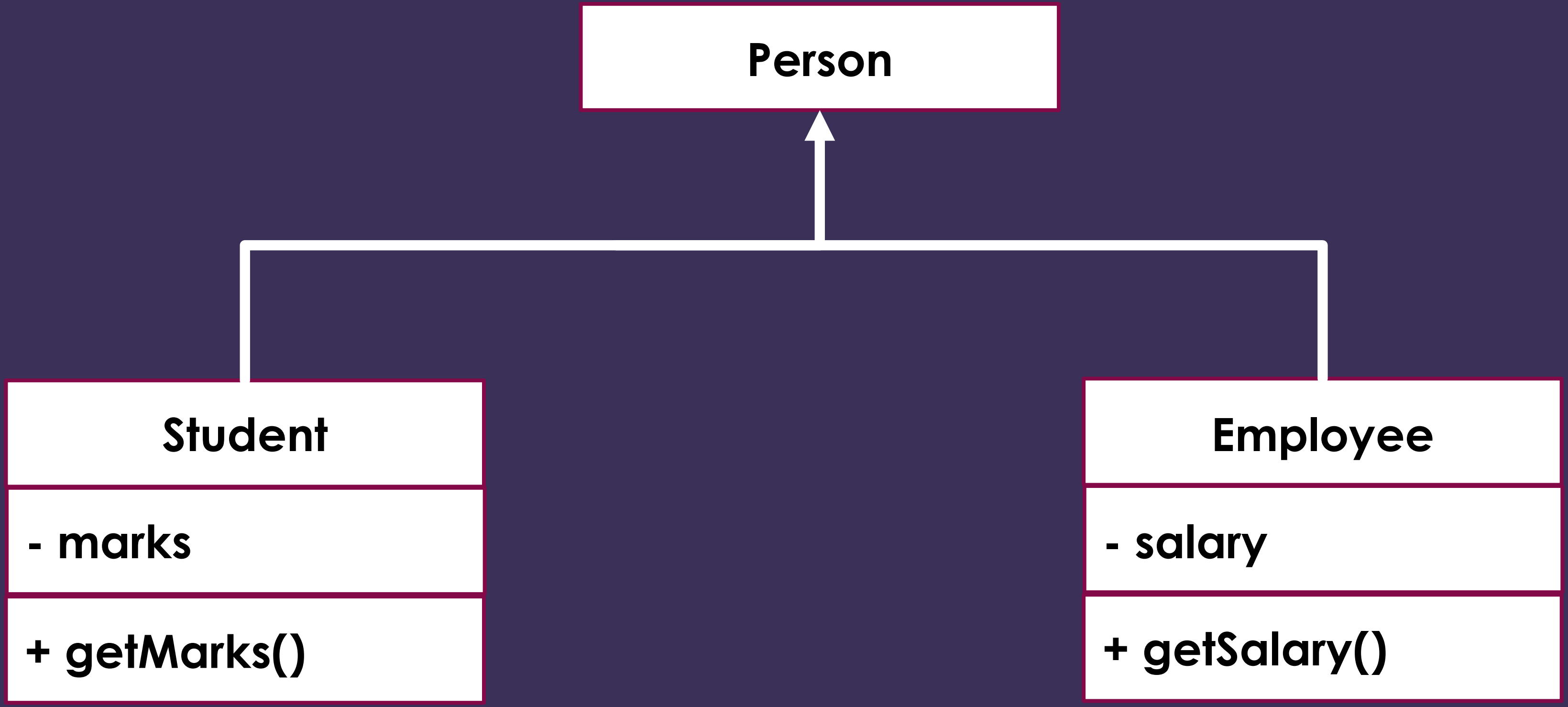
# Class Diagram

- Class name: Person
- Attributes: name, address, and birthdate
- Operations: getName(), getAddress(), getBirthDate(), and display()
- + denotes public members
- # denotes protected members
- - denotes private members

# UML Relationships

- Several relationship types can be encoded in UML
- A relationship is represented as lines with arrows
- Different arrowheads have different meanings
- Example relationship types are *inheritance* and *association*
- Inheritance represents a hierarchy between classes
- Association represents relationships between objects

# Class Diagram



# Instantiation

- To instantiate class you can use `new` operator
- For instance, `Student stud = new Student();`

  ```
  Student[] students = new Student[10];
  ```
- That creates the array only without the inner content
- To create the actual Students, you should do the following:

  ```
  for(int i=0; i<10; i++){
   students[i] = new Student();
  }
  ```

Thank You!