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
- UML
- 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);
```

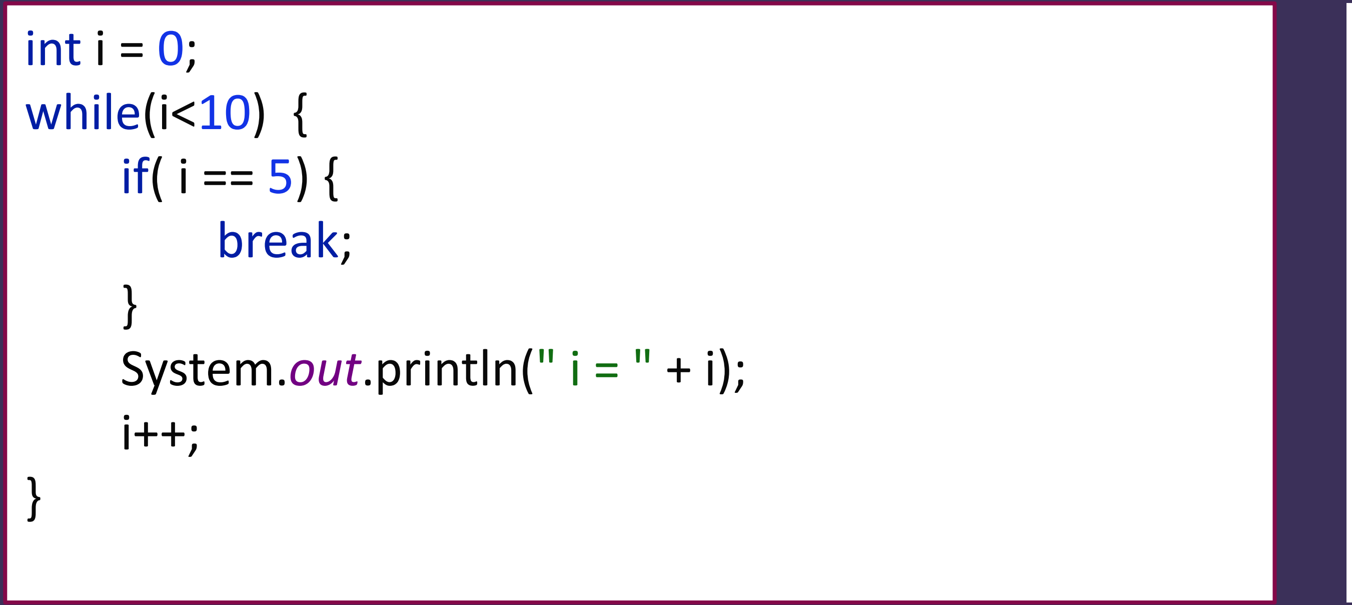
Loop Control

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

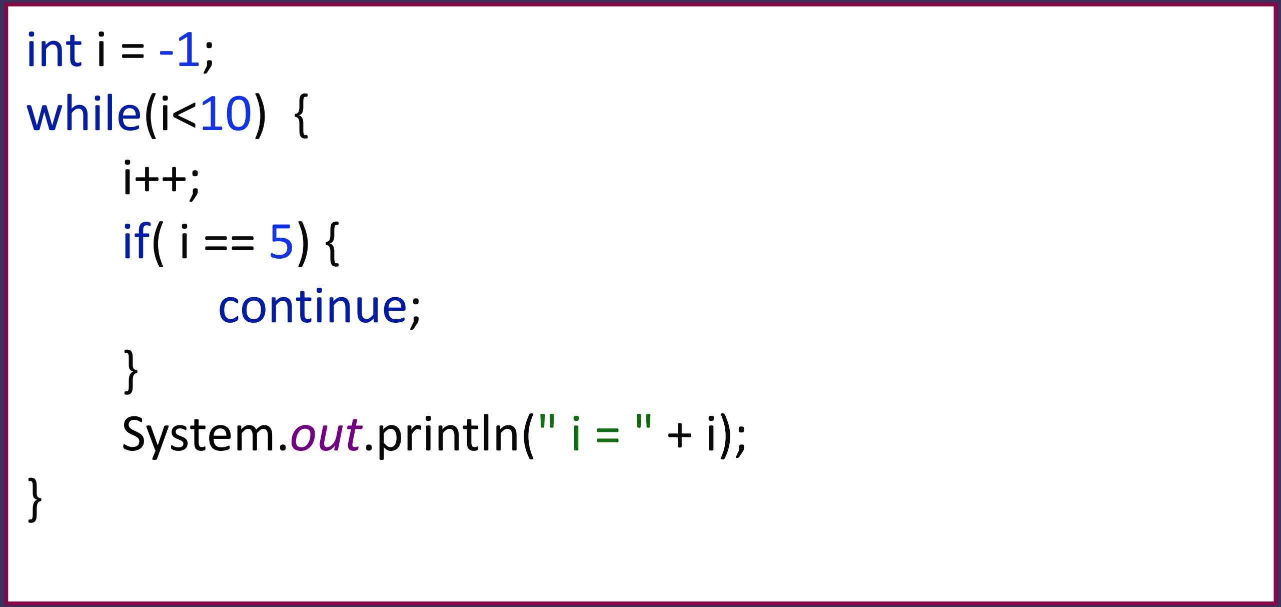
- What is the output?



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

Loop Control

- What is the output?



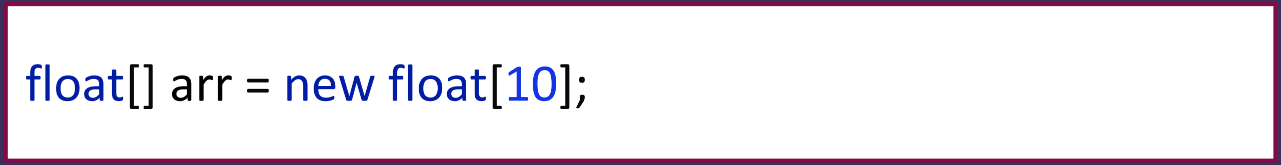
```
int i = -1;
while(i<10)  {
      i++;
      if( i == 5) {
            continue;
      }
      System.out.println(" i = " + i);
}
```

Arrays

Arrays

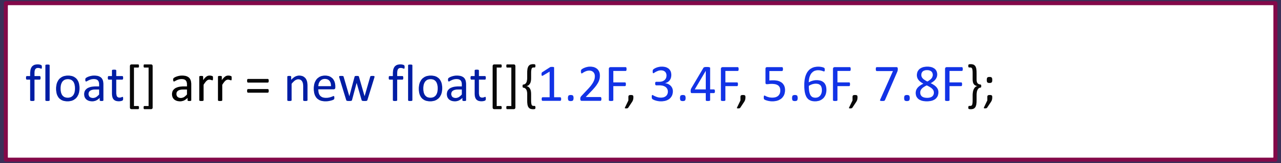
- Is simply a collection of items of the same type
- Has a fixed size
- Can hold any type, including simple types, e.g. int and float, or complex types, e.g. Student or Car
- Only holds references, i.e. does not hold the actual objects
- If any of position is not initialized, it is <u>*NULL*</u>

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 a 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

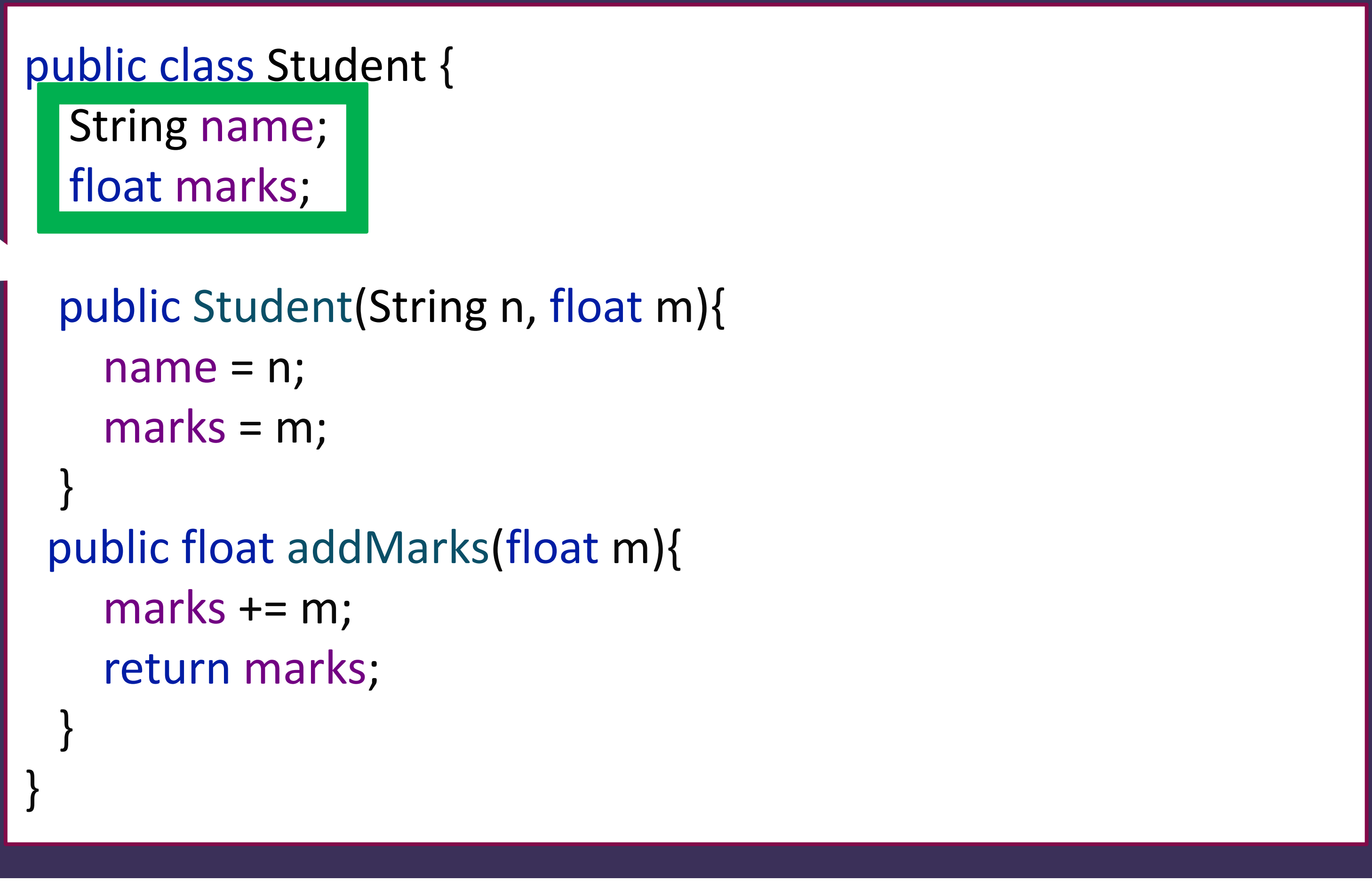
Class Name

```
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 an 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)

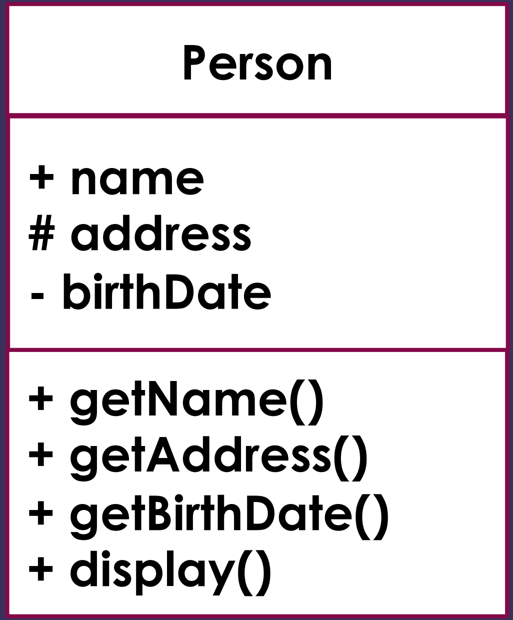
Unified Modeling Language (UML)

- 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
# address
- birthDate

+ getName()
+ getAddress()
+ getBirthDate()
+ display()

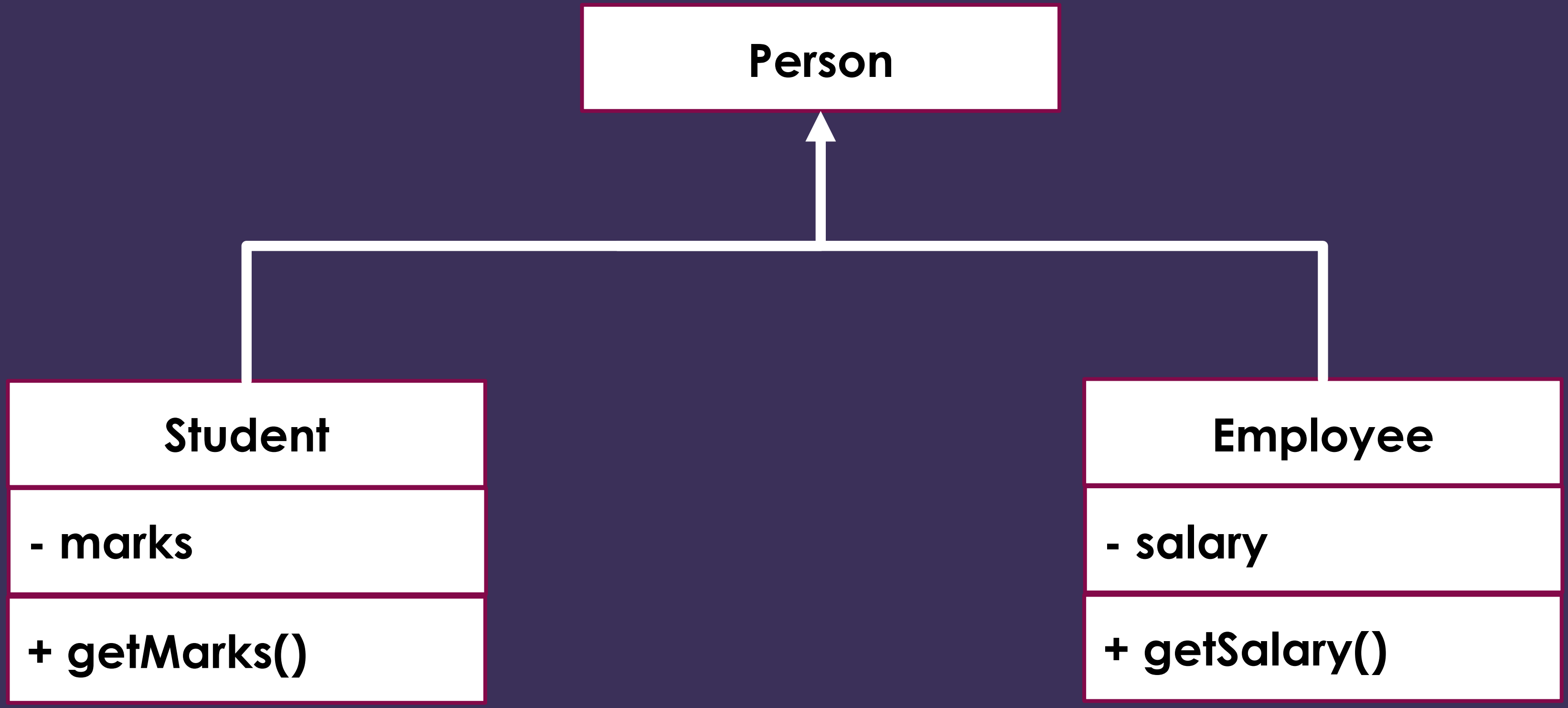
Class Diagram

- 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();
  }
  ```

Practice

- Create a UML diagram to represent Course
- Define attributes and methods of class Course
- Create a UML diagram to represent Professor
- Define attributes and methods of class Professor
- What is the relationship between Course and Professor
- What are other classes you can think of for the entire system

Packages

- Is a container for related classes, e.g. `java.util` and `java.io`
- The package name normally looks like `domain.subdomain.subsubdomain……`
- For instance: `org.apache.commons`
- Try to always group your classes into packages and subpackages
- At the top of the class you can find the name of the package to which it belongs: `package org.apache.commons;`

Class Access Modifiers

- Allowed class access modifiers are `public` or none → no `private` or `protected` (except for inner classes)
- If none → it is called package-local, i.e. it is visible only within the same package
- Non-public classes are meant for internal use only → cannot be utilized by any external user

Field Access Modifiers

- Field declaration has the form: `Access-modifier <static><final> datatype fieldname;`
- Access modifier: `private`, `protected`, `public`, and none
- `static` means that it is for the entire class and not for a specific object
- `final` means that its value *CAN NEVER BE CHANGED*

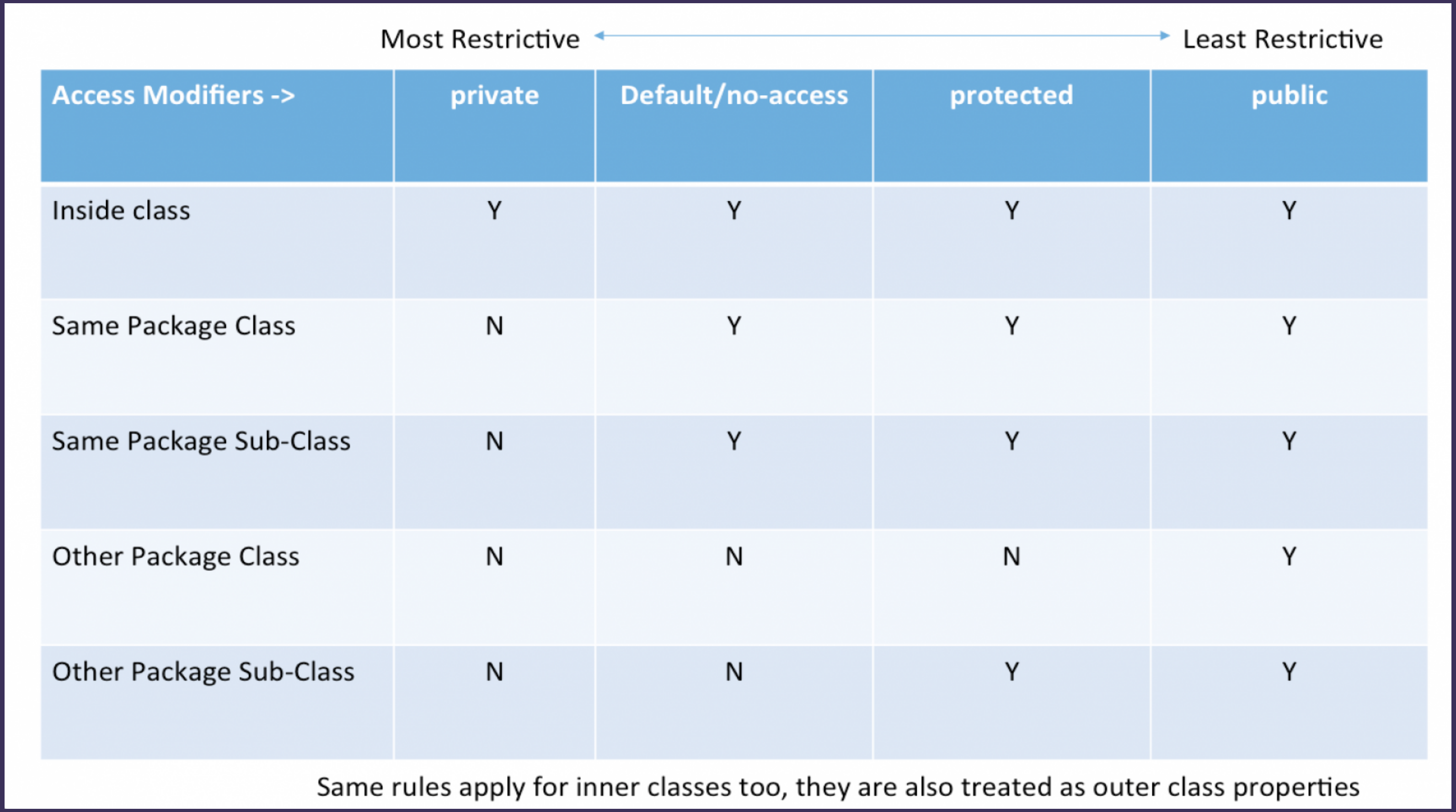
Field Access Modifiers

- `private` → visible within the same class only
- `None/package-local` → visible within the same class and classes withing the same package
- `protected` → visible within the same class, all child classes, and classes withing the same package
- `public` → visible anywhere

Field Access Modifiers

- `private` → visible within the same class only
- `None/package-local` → visible within the same class and classes withing the same package
- `protected` → visible within the same class, all child classes, and classes withing the same package
- `public` → visible anywhere

Field Access Modifiers



Most Restrictive ← → Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

Same rules apply for inner classes too, they are also treated as outer class properties

Person Class

```
public class Person {
    private String name;
    String address;
    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

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

Employee Class

```
public class Employee extends Person{
        public Employee(String name, String address) {
                super(name, address);
        }

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

Employee Class

```
public class Employee extends Person{
        public Employee(String name, String address) {
                        super(name, address);
        }

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

name is
invisible as it
has private
access

`static` Modifier

- Means that this field/method is not specific to an object
- It is for the entire class
- Can be accessed via class name, e.g. `Student.countOfStudents`, or via object name, e.g. `stud.countOfStudents`
- Non-static can ONLY be accessed via object name

Person Class

```
public class Person {
        public static int globalId;
        public int localId;
        private String name;
         private String address;

        public Person(String name, String address) {
                this.name = name;    this.address = address;
        }
}
```

Person Class

```
public class Person {
        public static int globalId;
        public int localId;
        private String name;
         private String address;

        public Person(String name, String address) {
                this.name = name;    this.address = address;
        }
}
```

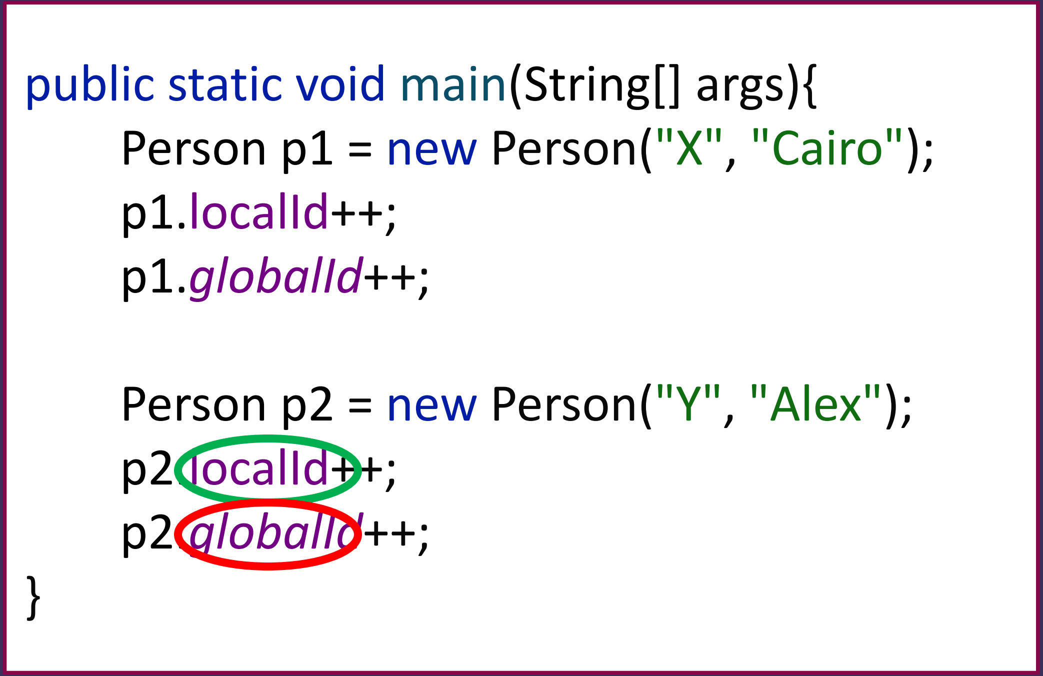
Main Class

```
public static void main(String[] args){
        Person p1 = new Person("X", "Cairo");
        p1.localId++;
        p1.globalId++;




}
```

Main Class



```
public static void main(String[] args){
        Person p1 = new Person("X", "Cairo");
        p1.localId++;
        p1.globalId++;

        Person p2 = new Person("Y", "Alex");
        p2.localId++;
        p2.globalId++;
}
```

`final` Modifier

- When used with field → the value of the field can never be modified once set
- Similar to `const` in C++
- `final` field must be initialized when declared or in the constructor
- For instance,

  ```
  public final int baseSalary = 1000;
  ```
- Can also be applied to methods and classes → to be discussed later

Thank You!