SENG 201 – Object Oriented Programming

2025 – 2026 Fall Lectures – Week 10

Ömer Mintemur

List of Objects

```java
public class Person {

    private String name;
    private String surname;
    private int age;

    Person(){

    }

    Person(String name_p, String surname_p, int age_p)
    {
        name = name_p;
        surname = surname_p;
        age = age_p;
    }
    public void showInfo()
    {
        System.out.println("Name is " + name +
                " Surname " + surname + " Age " + age);
    }
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person[] persons = new Person[10];
    persons[0].showInfo();
```

Null Pointer Exception

This creates an array of `Person` references, but the array elements are initially set to `null`. There are no actual `Person` objects in the array at this point–just `null` references.

List of Objects

```java
public class Person {

    private String name;
    private String surname;
    private int age;

    Person(){

    }

    Person(String name_p, String surname_p, int age_p)
    {
        name = name_p;
        surname = surname_p;
        age = age_p;
    }
    public void showInfo()
    {
        System.out.println("Name is " + name +
                " Surname " + surname + " Age " + age);
    }
```

WRONG !!

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person[] persons = new Person[10];
    persons[0].showInfo();
```

SOLUTION

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person[] persons = new Person[10];
    persons[0] = new Person("John","Sun",20);
    persons[0].showInfo();
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person[] persons = {new Person(), new Person("John", "Sun", 20)};

}
```

Static in Class

```java
public class Person {

    private String name;
    private String surname;
    private int age;

    private static int object_count = 0;

    Person(){
        object_count++;
    }

    Person(String name_p, String surname_p, int age_p)
    {
        name = name_p;
        surname = surname_p;
        age = age_p;
        object_count++;
    }
    public void showInfo()
    {
        System.out.println("Name is " + name +
                " Surname " + surname + " Age " + age);
    }
    public static int getObjectCount()
    {
        return object_count;
    }
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person p1 = new Person();
    Person p2 = new Person();
    Person p3 = new Person();
    System.out.println(p1.getObjectCount());
    System.out.println(p2.getObjectCount());
    System.out.println(p3.getObjectCount());
    System.out.println(Person.getObjectCount());

}
```

Static Variable:

- A static variable belongs to the class, not to any specific object
- All instances of the class share the same static variable

Static Methods:

- A static method can access and modify static variables
- Static methods cannot access instance variables or instance methods directly because they do not have a reference to a specific object.

This Keyword

```java
public class Cat {

    private String name;
    private int age;
    private String type;

    // Default Constructor
    Cat()
    {

    }
    Cat (String name, int age, String type)
    {
        name = name;
        age = age;
        type=type;
    }

    public void show()
    {
        System.out.println("Cat name is " + name +
                "\nAge is " + age + "\nType is " + type);
    }
```

```java
    public static void main(String[] args) {
        // TODO code application logic here
        Cat cat_1 = new Cat("John", 2, "Scottish");
        cat_1.show();
    }

}
```

```
run:
Cat name is null
Age is 0
Type is null
BUILD SUCCESSFUL
```

This Keyword

```java
public class Cat {

    private String name;
    private int age;
    private String type;

    // Default Constructor
    Cat()
    {

    }
    Cat (String name, int age, String type)
    {
        this.name = name;
        this.age = age;
        this.type=type;
    }

    public void show()
    {
        System.out.println("Cat name is " + name +
        "\nAge is " + age + "\nType is " + type);
    }
```

This keyword refers to object itself in a class

```java
    public static void main(String[] args) {
        // TODO code application logic here
        Cat cat_1 = new Cat("John", 2, "Scottish");
        cat_1.show();
    }

}
```

```
run:
Cat name is John
Age is 2
Type is Scottish
BUILD SUCCESSFUL
```

This Keyword

- This keyword refers to the current object in a method or constructor. It allows access to the instance variables and the methods of the class
- When a constructor or method has parameters with the same name as instance variables, the this keyword is used to distinguish the instance variables from the parameters.

This Keyword – Compare Two Cats

```java
public static void main(String[] args) {
    // TODO code application logic here
    Cat cat_1 = new Cat("John", 2, "Scottish");
    Cat cat_2 = new Cat("Mike", 1, "British");

    // Is the type of cat_1 is equal to cat_2

    if (cat_1.getType()==cat_2.getType())
    {
        System.out.println("Cats' type are equal");
    }
    else
    {
        System.out.println("Cats' type are not equal");
    }
```

```java
public class Cat {

    private String name;
    private int age;
    private String type;
    // Default Constructor
    Cat(){}
    Cat (String name, int age, String type)
    {
        this.name = name;
        this.age = age;
        this.type=type;
    }
    public void show()
    {
        System.out.println("Cat name is " + name +
        "\nAge is " + age + "\nType is " + type);
    }
    public String getType()
    {
        return this.type;
    }
```

This Keyword – Compare Two Cats (V2)

```
public static void main(String[] args) {
    // TODO code application logic here
    Cat cat_1 = new Cat("John", 2, "Scottish");
    Cat cat_2 = new Cat("Mike", 1, "British");

    // Is the type of cat_1 is equal to cat_2
    cat_1.compareCat(cat_2);

}


// Is the type of cat_1 is equal to cat_2
//cat_1.compareCat(cat_2);
cat_2.compareCat(cat_1);
```

```
public class Cat {

    private String name;
    private int age;
    private String type;
    // Default Constructor
    Cat(){}
    Cat (String name, int age, String type)
    {
        this.name = name;
        this.age = age;
        this.type=type;
    }
    public void show()
    {
        System.out.println("Cat name is " + name +
        "\nAge is " + age + "\nType is " + type);
    }
    public void compareCat(Cat c)
    {
        if(this.type == c.getType())
        {
            System.out.println("Cats' type are equal");
        }
        else
        {
            System.out.println("Cats' type are not equal");
        }
    }
    public String getType()
    {
```

This Keyword

```java
public static void main(String[] args) {
    // TODO code application logic here
    Cat cat_1 = new Cat("John", 2, "Scottish");
    Cat cat_2 = new Cat("Mike", 1, "British");

    Cat cat_3 = cat_1.returnCat();
    cat_3.show();

}
```

```java
public class Cat {

    private String name;
    private int age;
    private String type;
    // Default Constructor
    Cat(){}
    Cat (String name, int age, String type)
    {
        this.name = name;
        this.age = age;
        this.type=type;
    }
    public void show()
    {...4 lines }
    public void compareCat(Cat c)
    {...10 lines }
    public Cat returnCat()
    {
        return this;
    }
    public String getType()
    {...3 lines }
```

This Keyword

```java
public class Trick {
    private int x;
    private int y;

    // Constructor
    Trick(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Method to update 'x'
    Trick setX(int x) {
        this.x = x;
        return this;
    }

    // Method to update 'y'
    Trick setY(int y) {
        this.y = y;
        return this;
    }

    // Method to display the values
    void display() {
        System.out.println("x = " + this.x + ", y = " + this.y);
    }
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Trick obj = new Trick(1, 2);

    obj.setX(10).setY(20).display();

    Trick obj2 = new Trick(5, 5);
    obj2.display();

}
```

This Keyword

```java
public class TrickMore {
    private int x;

    TrickMore(int x) {
        this.x = x;
    }

    void updateValue(int x) {
        x = x + 10;
    }

    void correctUpdate(int x) {
        this.x = x + 10;
    }

    void display() {
        System.out.println("x = " + this.x);
    }
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    TrickMore obj = new TrickMore(5);

    obj.updateValue(10);
    obj.display();

    obj.correctUpdate(10);
    obj.display();

}
```

Inheritance

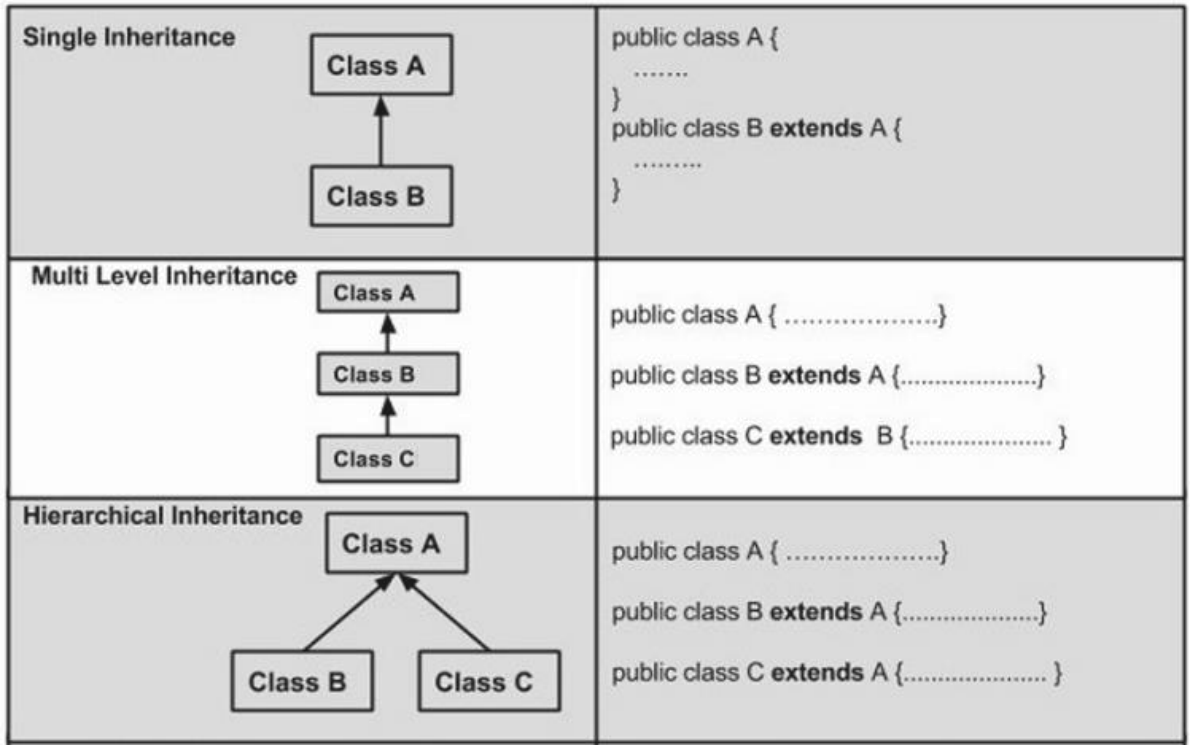
- What is Inheritance?
  - Inheritance is a mechanism in OOP where a class (child/subclass) derives properties and behaviors (attributes and methods) from another class (parent/superclass)
- Enable code reusability, modularity and hierarchy representation.

Inheritance

- How it works?
  - The extends keyword is used to establish inheritance
  - A subclass inherits all **public** members of its parent class
- **Private** members of the parent are not directly accessible but can be accessed through getters/setters or protected methods

Inheritance

- **Single Inheritance**: A class inherits from one parent class.
- **Multilevel Inheritance**: A class inherits from a class, which itself inherits from another class
- **Hierarchical Inheritance**: Multiple subclasses inherit from a single parent class
- **Multiple Inheritance is not available in Java**

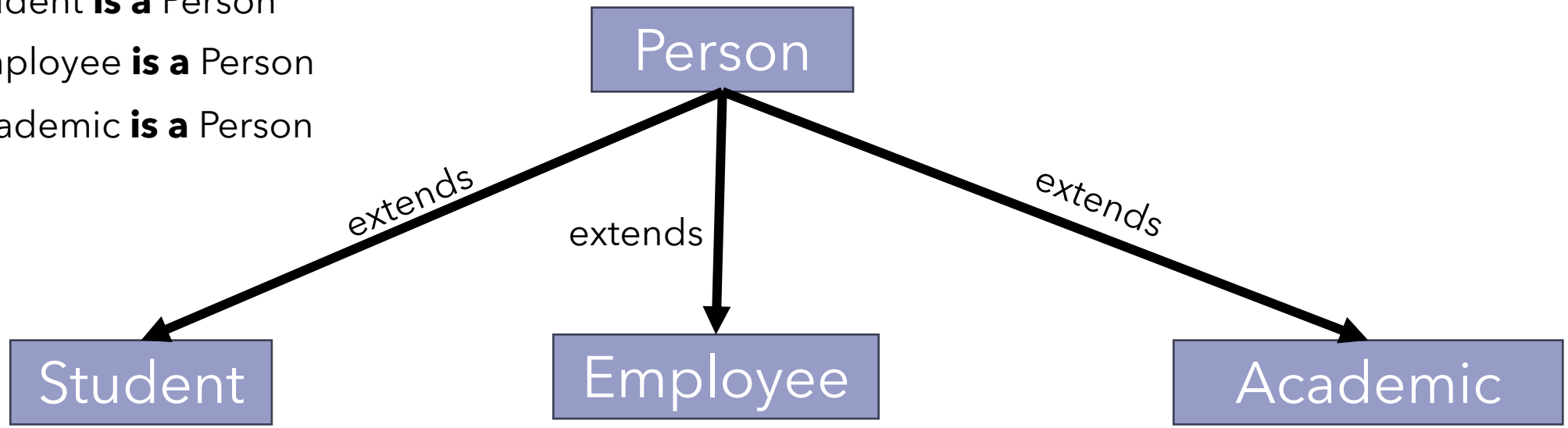


Inheritance

Each Student **is a** Person

Each Employee **is a** Person

Each Academic **is a** Person



*Use inheritance when there is a clear **"is-a" relationship** between classes*

Inheritance

```java
public class Person {
    Person(){}

    public void sayHi()
    {
        System.out.println("I am a Person");
    }
```

```java
public class Student extends Person{

    Student(){}

}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person p1 = new Person();
    p1.sayHi();
    Student s1 = new Student();
    s1.sayHi();
```

```
run:
I am a Person
I am a Person
```

Inheritance - Polymorphism

```java
public class Person {
    Person(){}

    public void sayHi()
    {
        System.out.println("I am a Person");
    }
```

```java
public class Student extends Person{

    Student(){}

    public void sayHi()
    {
        System.out.println("I am a Student");
    }
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Person p1 = new Person();
    p1.sayHi();
    Student s1 = new Student();
    s1.sayHi();
```

```
run:
I am a Person
I am a Student
```

A subclass provides a ***specific implementation of a method defined in its parent class. If not overridden, all public methods in extended class can be used in subclass.***

Inheritance

```java
public class Person {
    public String name;
    public String surname;
    public int age;
    Person(){}

    public void sayHi()
    {
        System.out.println("I am a Person");
    }

public class Student extends Person{

    Student(){}

    public void sayHi()
    {
        System.out.println("I am a Student");
    }
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    Student s1 = new Student();
    s1.
```

age	int
name	String
surname	String
equals(Object obj)	boolean
getClass()	Class<?>
hashCode()	int
notify()	void
notifyAll()	void
sayHi()	void
toString()	String
wait()	void
wait(long timeoutMillis)	void
wait(long timeoutMillis, int nanos)	void

Inheritance

```java
public class Person {
    private String name;
    private String surname;
    private int age;
    Person(){}

    Person(String name, String surname, int age)
    {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }

    public void sayHi()
    {
        System.out.println("I am a Person");
    }
```

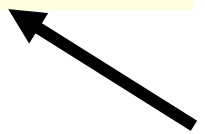
```java
public class Student extends Person{

    private int studentId;

    Student(){}
    Student(int studentId)
    {
        this.studentId = studentId;
    }

    public void sayHi()
    {
        System.out.println("I am a Student");
    }

  public static void main(String[] args) {
      // TODO code application logic here
      Student s1 = new Student(123);
```



Since every student is a Person, each Student should have name, surname, and age. Moreover, every student should have student id.

Inheritance

```java
public class Person {
    private String name;
    private String surname;
    private int age;
    Person(){}

    Person(String name, String surname, int age)
    {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }
    public void sayHi()
    {
        System.out.println("I am a Person");
    }
    public String showInfo()
    {
        return "Name is " + this.name + " Surname is "
        +this.surname+" Age is "+this.age;
    }
```

```java
public class Student extends Person{

    private int studentId;

    Student(){}
    Student(String name, String surname, int age, int studentId)
    {
        super(name,surname,age);
        this.studentId = studentId;
    }
    public void sayHi()
    {
        System.out.println("I am a Student");
    }
    public String showInfo()
    {
        return super.showInfo() +
        " Student ID is " + this.studentId;

    }
```

Inheritance (Not extendable)

```java
public final class Person {
    private String name;
    private String surname;
    private int age;
    Person(){}

    Person(String name, String surname, int age)
    {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }
```

You can not extend a class if it is defined as final

```java
public class Person {
    private String name;
    private String surname;
    private int age;
    Person(){}

    Person(String name, String surname, int age)
    {
        this.name = name;
        this.surname = surname;
        this.age = age;
    }
    public final void sayHi()
    {
        System.out.println("I am a Person");
    }
    public String showInfo()
    {
        return "Name is " + this.name + " Surname is "
        +this.surname+" Age is "+this.age;
    }
```

You can not override a method if it is defined as final

Inheritance (Ordering)

```java
public class Person {

    Person(){
        System.out.println("Person constructor");
    }
}
```

```java
public class Student extends Person{

    Student(){
        System.out.println("Student Constructor");
    }

}
```

```java
public class JuniorStudent extends Student {

    JuniorStudent()
    {
        System.out.println("Junior Student Constructor");
    }
}
```

```java
public static void main(String[] args) {
    // TODO code application logic here
    JuniorStudent j1 = new JuniorStudent();
}
```

```
run:
Person constructor
Student Constructor
Junior Student Constructor
```