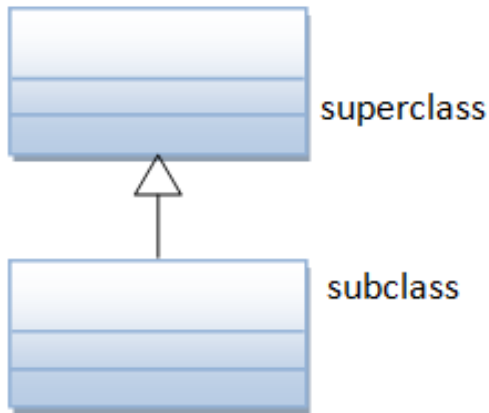




Inheritance



Dr. Abdelkarim Erradi
CSE@QU

Outline

- ① Inheritance Basics
- ② Overriding

Some slides are based on the textbook *Java™ How to Program, 11e*

Inheritance

- **Ideas**

- Common attributes and methods are placed in a **superclass** (also called *parent class* or *base class*)

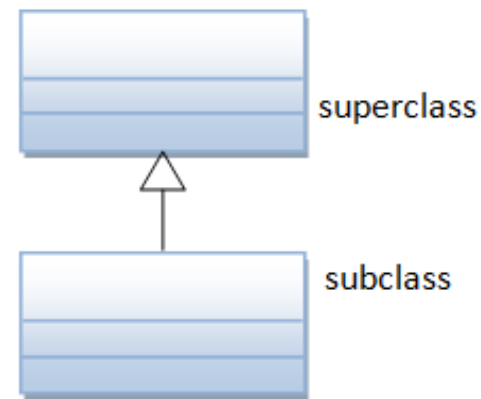
- You can create a subclass that **inherits**

Attributes and methods of the super class

- Subclass also called *child class* or *derived class*

- Subclass has access to all **non-private** (i.e., *public* and *protected*) **attributes and methods of the superclass**

- Subclass can extend the superclass by **adding new attributes/methods** and/or **overriding the superclass methods**



- **Syntax**

- public class SubClass **extends** SuperClass { ... }

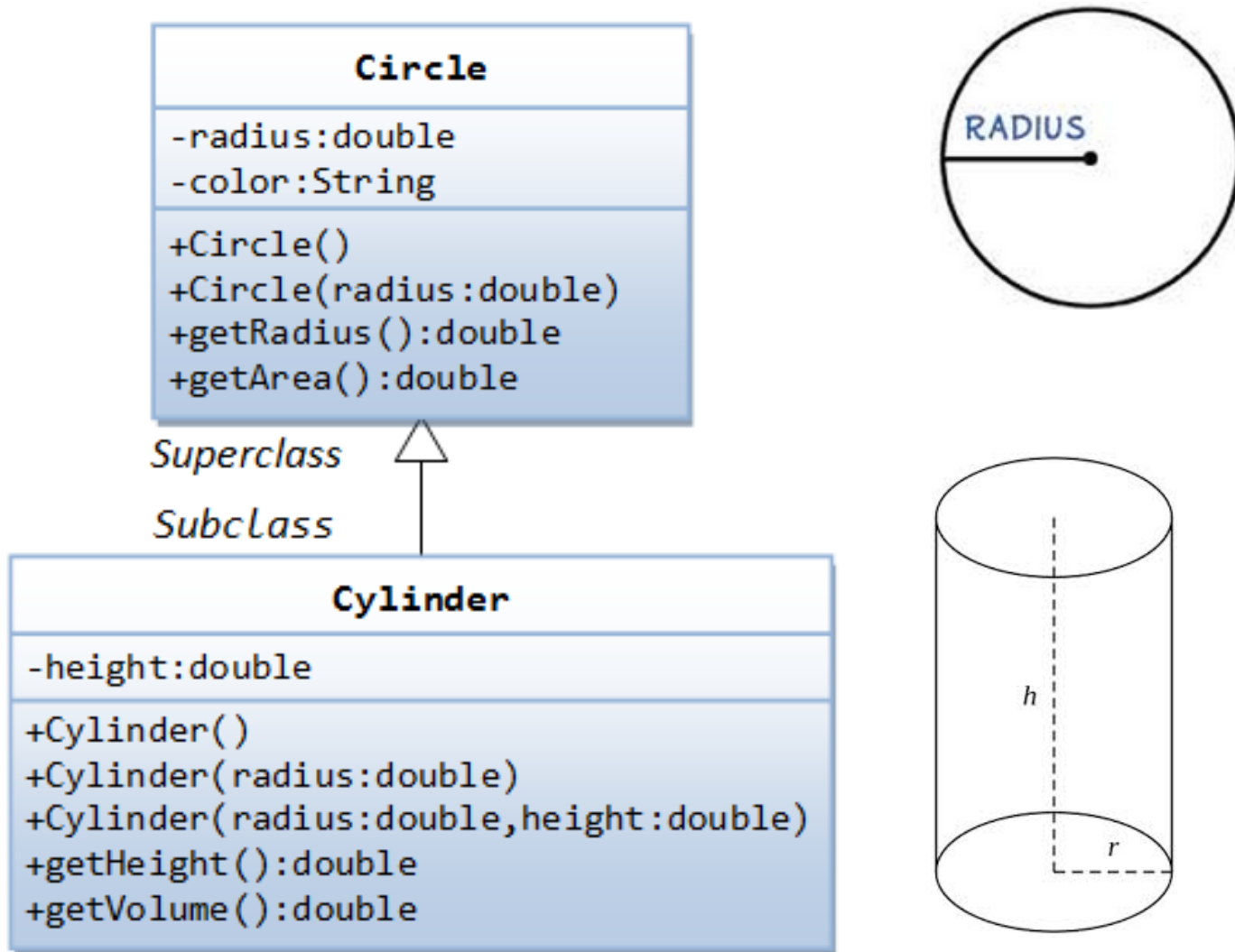
- **Motivation**

- Supports the key OOP goal of **code reuse**. Allow us to design **class hierarchies** so that **shared behavior is placed in a super class** then inherited by subclasses (i.e., avoids writing the same code twice to ease maintenance)

Benefits of Inheritance

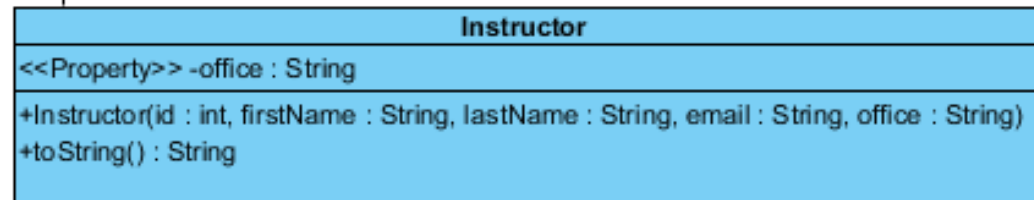
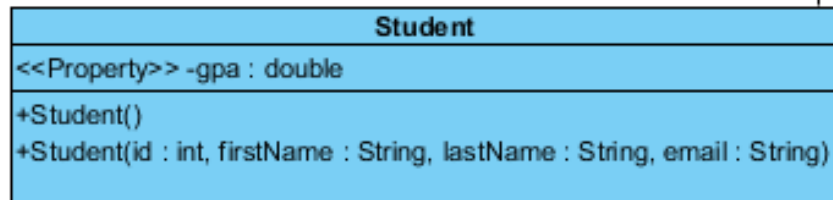
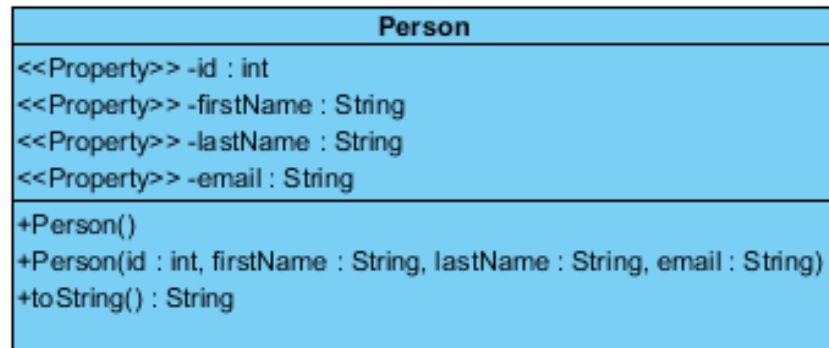
- Benefits of inheritance
 - Can save time during program development by basing new classes on existing tested and quality classes.
 - Reduces duplication => eases maintainability of the code
 - Allow us to **avoid the “copy-and-paste” approach** which **spreads copies of the same code throughout a system, creating a code-maintenance nightmare**
 - Localizing the effects of changes is a good software engineering practice. Changes are made once for common attributes/methods in the superclass, subclasses then inherit the changes.
- **Limitation:**
 - Java supports only **single inheritance**, in which each class is derived from exactly one direct superclass

Inheritance Example



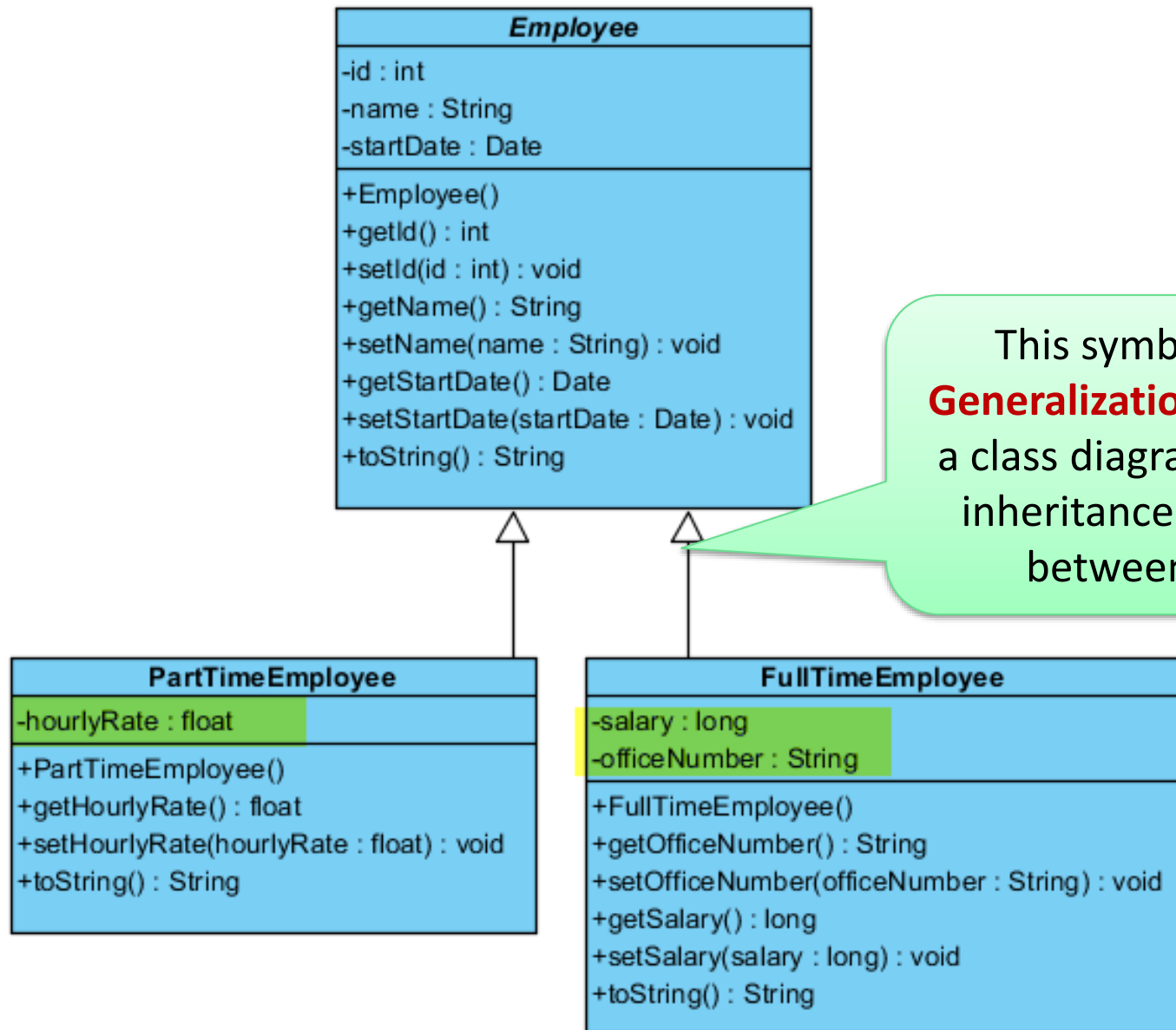
See implementation in ***inheritance.circle*** package

Person Example



- The Person class has the common attributes and methods
- Each subclass can add its own specific attributes and methods (e.g., **office** for Instructor and **gpa** for Student)
- Each subclass can **override** (redefine) the parent method (e.g., Instructor class overrode the toString() method).

Another Example - Employee Hierarchy



This symbol is called **Generalization**. It is used in a class diagram to indicate inheritance relationship between classes.

More Examples

Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount

is-a relationship **vs.** has-a relationship

- We distinguish between the **is-a relationship** and the **has-a relationship**
- ***Is-a*** represents inheritance
 - In an *is-a* relationship, an object of a subclass can also be treated as an object of its superclass
 - e.g., Student is a Person
- ***Has-a*** represents composition
 - In a *has-a* relationship, an object contains as attributes references to other objects
 - E.g., Student has a list of courses

The Object Class

- **Object** is the **root** class of all classes in Java
- All other classes are descendents of **Object**
- **Object** is part of the **java.lang** package
- Useful Object methods:
 - **toString** - returns a **string representation** of the object (by default, its class name and id, but this can be overridden).
 - **equals** - tests for equality of value of two different objects
 - **getClass** - returns the class to which an object belongs

toString method

- `toString` is one of the methods that every class inherits from the `Object` class
 - Returns a string representation of the object.
 - Called implicitly whenever an object must be converted to a `String` representation.
- By default `toString` method returns a `String` that includes the name of the object's class
 - It can be overridden by a subclass to specify an appropriate `String` representation

```

public class Circle {
    /** The radius of the circle */
    private double radius;

    /** Construct a circle with radius 1 */
    public Circle() {
        radius = 1.0;
    }

    /** Construct a circle with a specified radius */
    public Circle(double newRadius) {
        radius = newRadius;
    }

    /** Return radius */
    public double getRadius() {
        return radius;
    }

    /** Set a new radius */
    public void setRadius(double newRadius) {
        radius = (newRadius >= 0) ? newRadius : 0;
    }

    /** Return the area of this circle */
    public double findArea() {
        return radius * radius * 3.14159;
    }

    public String toString() {
        return "This is a circle";
    }
}

```

Even though there is not extends keyword, Circle is a subclass of Object.

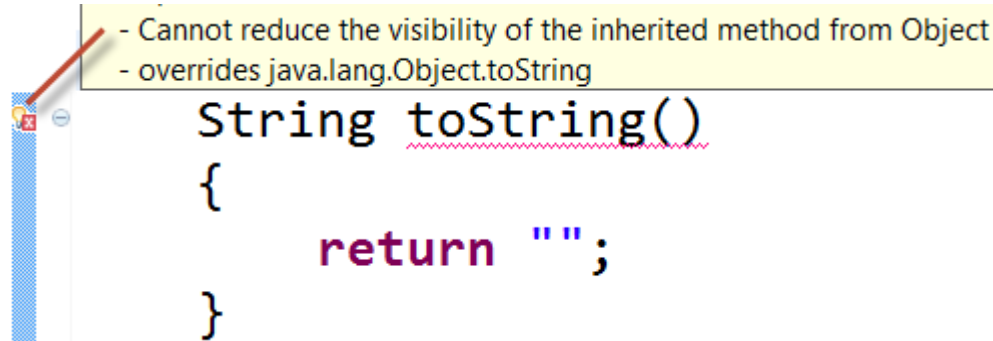
Object is the root of ALL classes in Java

Overriding the Object class's toString method

Overriding

Overriding

- To override a superclass method, a subclass must **declare a method with the same signature as the superclass method**
 - Same access modifier should be used. E.g. if the superclass method is public the overridden method should also be public.



The screenshot shows a code editor with a Java method `String toString()` being overridden. A yellow tooltip box points to the method signature with the following text:

- Cannot reduce the visibility of the inherited method from Object
- overrides java.lang.Object.toString

```
String toString()  
{  
    return "";  
}
```

- **@Override** is a optional annotation
 - Declare overridden method with the *@Override* annotation to ensure at compilation time that you defined their signatures correctly.
 - It's always better to find errors at compile time rather than at runtime.

Overriding

- Overriding allow the subclass to replace/extend the behavior of the superclass.
- When a subclass method overrides an inherited superclass method, the superclass method can be accessed from the subclass by preceding the superclass method name with keyword **super** and a dot (.) separator.

```
public class Instructor extends Person {  
    private String office;  
  
    public String toString() {  
        return super.toString() + " - Office: " + office;  
    }  
}
```

Using **@Override** is optional

- Child class (mistake!)

```
public class Circle extends Shape{  
    //typo in method name  
    public double getarea() { ... }  
}
```

This code will compile, but when you call `getArea` at runtime, you will get the version from `Shape`, since there was a typo in this name (lowercase a)

- Catching such mistake at compile time

```
public class Circle extends Shape {  
    @Override  
    public double getarea() { ... }  
}
```

This tells the compiler “I am overriding a method from the parent class”. If there is no such method in the parent class, the code won’t compile. **@Override** is recommended but optional.

Inheritance and Constructors

- **Constructors are not inherited**
- But a subclass constructor **can call its direct superclass's constructor** to initialize the instance variables inherited from the superclass

```
public Instructor(int id, String firstName, String lastName,  
                  String email, String office) {  
    //Explicit call to superclass constructor  
    super(id, firstName, lastName, email);  
    this.office = office;  
}
```

- If a subclass constructor does not include an explicit call to the superclass constructor, Java implicitly calls the superclass's default constructor.

```
public Instructor() {  
    //The superclass default constructor will be implicitly called  
}
```

Constructors in Subclasses

- Instantiating a subclass object begins a **chain of constructor calls**
 - The subclass constructor, before performing its own tasks, invokes its direct superclass's constructor
- If the superclass is derived from another class, the superclass constructor invokes the constructor of the next class up the hierarchy all the way back to the Constructor of Object
- The last constructor called in the chain is always class Object's constructor.
- Each superclass's constructor initialized the superclass attributes that the subclass object inherits.



Software Engineering Observation

Java ensures that even if a constructor does not assign a value to an instance variable, the variable is still initialized to its default value (e.g., 0 for primitive numeric types, false for booleans, null for references).

Summary

- Inheritance = placing common attributes and methods in a superclass so that subclasses can reuse them
- Subclass extends a superclass and inherits the superclass's members, though the **private** superclass members are hidden from the subclass
- Subclass can include additional members or override superclass methods
- Subclass can override superclass methods
- **Constructors are not inherited** but a subclass constructor **can call its direct superclass's constructor** to initialize the superclass attributes