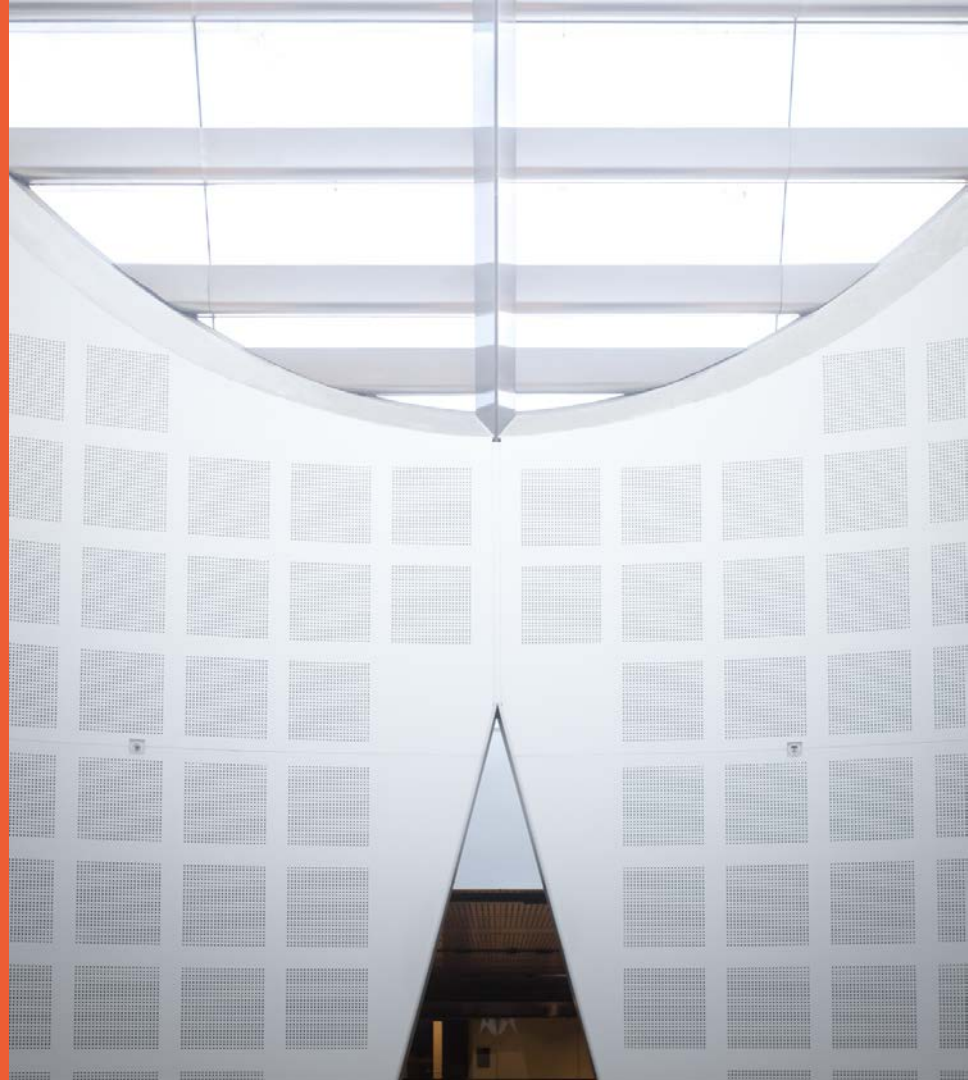
# COMP9103: Software Development in Java

## W6: Inheritance & Polymorphism

**Presented by**
Dr Ali Anaissi
School of Computer Science

THE UNIVERSITY OF SYDNEY



# Introduction to Inheritance

# Introduction to Inheritance

- ***Inheritance*** **is a mechanism by which a new class is derived from an existing class.**
  - The existing class that depicts common and general fields and methods is called as the **base class**, or **superclass**
    - E.g. the ***Person*** class
  - The classes defined as **extensions** of the superclass to **inherit all the fields and methods from the superclass** are called as **subclasses**, or **derived classes**
    - E.g. we define a ***Student*** class on the basis of ***Person*** class, then ***Student*** class is a subclass of ***Person*** class
- Inheritance is a powerful way to support **software reuse**.

# Introduction to Inheritance

- ***Superclass – subclass*** inheritance implies an "***is-a***" relationship
  - A subclass object "*is-a*" a superclass object
  - The superclass ***generalizes*** subclasses

- Examples:
  - ***Student*** "*is-a*" ***Person***
  - ***ChequeAccount*** "*is-a(n)*" ***Account***

- In general, a subclass ***is a special type of*** superclass.

# Code-reuse in Java

- You write code for a superclass, and others *reuse* it in the subclasses, without re-writing from scratch
  - We use java keyword **extends** to define the inheritance

- **In Java, a class directly extends only one superclass**
  - It implicitly extends the ***Object*** class if nothing else is declared.

- The subclass inherits all fields and methods of the parent.

# Example: class inheritance with *extends*

```
public class Customer {

   private double creditCardBalance;
   private double chequeAccountBalance;
   private String name;
...
```

superclass

***YoungCustomer*** has **all** the fields and methods of **Customer**, plus a few of its own.

***YoungCustomer* "*is-a*" *Customer***

```
class YoungCustomer extends Customer {
   private int age;
...
```

subclass

# Access modifiers

- *Access modifiers* **control the accessibility/visibility** of fields and methods in a Java class
  - [when there is no modifier stated, it is ***default*** access]
  - ***public***
  - ***private***
  - ***protected***
- If you want to **access** superclass members in the subclasses, you must make the superclass members as **protected**. But always make them private

# Inheritance: Fields

# Inheriting Fields

- Fields inheritance: all fields from the superclass are automatically inherited
  - Subject to access modifiers, **a subclass has no access to *private* fields of its superclass**
- You can add new fields in the subclass that are not defined in the superclass

```
public class Customer {

   private double creditCardBalance;
   private double chequeAccountBalance;
   private String name;
…
```

superclass

*YoungCustomer* has **all** the fields and methods of **Customer**, plus a few of its own.

*YoungCustomer* "*is-a*" *Customer*

```
class YoungCustomer extends Customer {
   private int age;
…
```

subclass

# Inheritance: Constructors

# Constructors and inheritance

– The first statement in any subclass constructor should be a call to a superclass constructor **super**(…)

```
class YoungCustomer extends Customer {
    private int age;

    public YoungCustomer(String name, double ccb, double cab, int age)
    {
            super(name,ccb,cab);
            this.age = age;
    }
……
```

# Inheritance: Methods

# Methods

– **How do we know which method to apply?**

***superclass***

```
public class Customer {
...
```

***subclass***

```
class YoungCustomer extends Customer {
…
```

***client class***

```
…
// calling a method from either subclass or superclass
…
```

# Inheriting Methods

- **Inherited methods:**
  - In this case, no new implementation of the superclass methods, and superclass methods can be applied to the subclass objects

*superclass*

```
public class Customer {
...
  public boolean inDebt() {
      return (wealth() < 0);
}
...
```

*subclass*

```
class YoungCustomer extends Customer {
... /* inherit inDebt() from super &
       no new implementation of inDebt() in the subclass*/
```

*another class*

```
...
YoungCustomer babyface = new YoungCustomer("Baby", 100, 100);
If (babyface.inDebt()) System.out.println("Bad Baby!");
...
```

*Superclass method is used*

# Adding Methods

- **Added methods:**
  - In subclass, define a new method that doesn't exist in the superclass
  - New methods can be applied only to subclass objects

*superclass*

```
public class Customer {
... // no method birthdayGift() defined
..
```

*subclass*

```
class YoungCustomer extends Customer {
...
  public void birthdayGift() {
     depositChequeAccount(10);
}
..
```

*another class*

```
...
YoungCustomer babyFace = new YoungCustomer("Baby", 100, 100);
if (babyFace.sameDay(Date.today)) babyFace.birthdayGift();
...
```

*Subclass method is used*

# Overriding Methods

- **Overridden methods:**
  - The subclass defines a **different implementation** of a superclass method
  - The subclass method **must have same signature (same name, same parameter types), same return type** as the superclass method

```
public class Customer {
...
  public void writeCheque(int amount) {
        chequeAccountBalance = chequeAccountBalance – amount - fee;
}
...
```

*superclass*

*Identical signature*

*Overwritten by subclass*

```
class YoungCustomer extends Customer {
   private int age;
   ...
  public void writeCheque(int amount) {
        chequeAccountBalance = chequeAccountBalance – amount ;
}
...
```

*subclass*

# Overriding Methods

- Overridden methods :
  - If the method is invoked from an object of the subclass type, then the overriding method is executed

```
public class Customer {
...
   public void writeCheque(int amount) {
            chequeAccountBalance = chequeAccountBalance – amount - fee;
}
...
```

```
class YoungCustomer extends Customer {
   private int age;
   ...
   public void writeCheque(int amount) {
         chequeAccountBalance = chequeAccountBalance – amount ;
}
...
```

subclass

```
YoungCustomer babyFace = new YoungCustomer("Baby", 100, 100)
babyFace.writeCheque(20);...
```

Subclass method is used

# Overriding Methods

- Note
  - We can use the superclass method in the subclass
    - Use "***super***": ***super.methodName(parameters)*** in subclass

```
class YoungCustomer extends Customer {
…
   public void writeCheque(int amount) {
       super.writeCheque(amount);
       chequeAccountBalance += fee;
}
…
```

```
…
YoungCustomer babyFace = new YoungCustomer("Baby", 100, 100)
babyFace.writeCheque(20);
…
```

# Aside: *toString* method

- Every class in java is child of Object class.
  - Object class contains toString() method.
- The toString() method is used to get a string representation of an object
- Whenever we try to print the Object reference then internally toString() method is invoked.
- If we did not define toString() method in your class then Object class toString() method is invoked.
- Otherwise our implemented/Overridden toString() method will be called.

# *toString* method

```
class Object {
…
        public String toString() {
          return getClass().getName()+"@"+Integer.toHexString(hashCode());
        }
…
```

```
public class Customer {
    private double creditCardBalance;
    private double chequeAccountBalance;
    private String name;
…}
```

```
public class CustomerTester {
 public static void main(String[] args) {
    Customer p = new Customer("Peter",-1276,423);
    Customer m = new Customer("Mary", -24, 165);
    System.out.println(p);
    System.out.println(m);
}
```

Prints out
***src5214.Customer@e48e1b***
***src5214.Customer@12dacd1***

# *toString* method

```
public class Customer {
   private double creditCardBalance;
   private double chequeAccountBalance;
   private String name;
. ... ..
public String toString(){
        return name + "\t" + creditCardBalance + "\t" +
                chequeAccountBalance;
}
.....
```

New **toString** method, overwrites Java supplied **toString** method

```
public class CustomerTester {
 public static void main(String[] args) {
    Customer p = new Customer("Peter",-1276,423);
    Customer m = new Customer("Mary", -24, 165);
    System.out.println(p);
    System.out.println(m);
}
```

Same test class

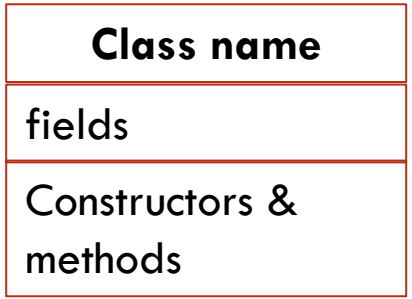
Prints out

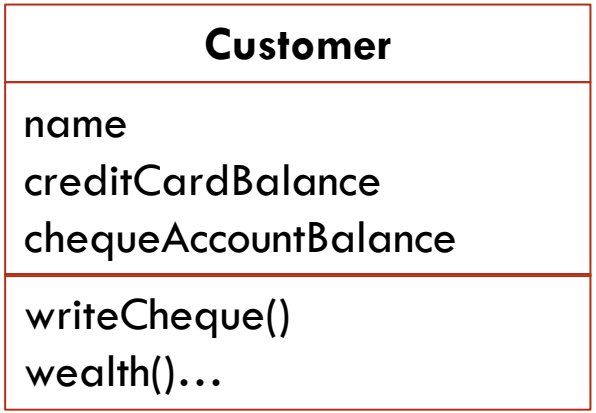
| | | |
|---|---|---|
| *Peter* | *-1276* | *423* |
| *Mary* | *-24* | *165* |

# UML
# (Unified Modeling Language)

# UML Diagrams

- UML (Unified Modeling Language) notation is for conceptual models, ideal for OO code structure!
  - Draw each class as a rectangle
  - We can add attributes and methods to them like this:

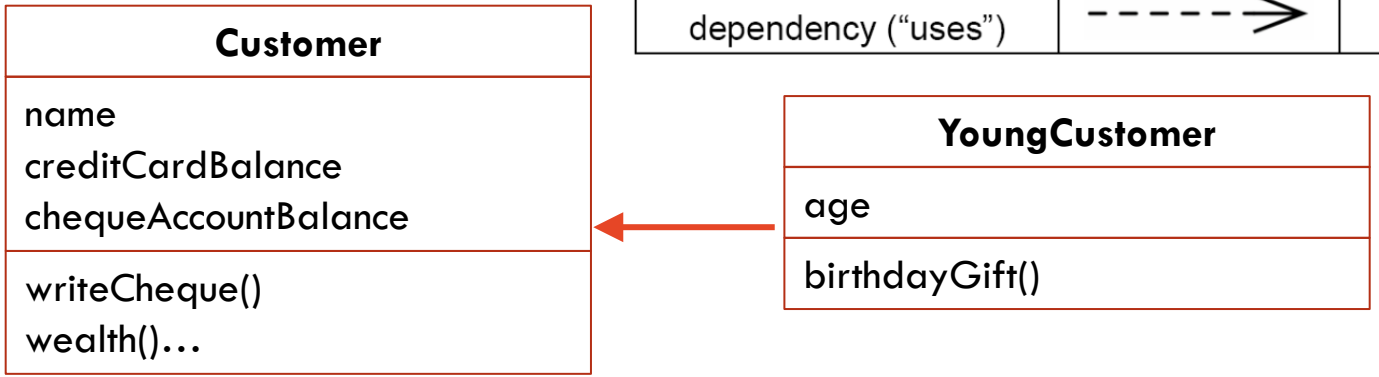


# UML Relations

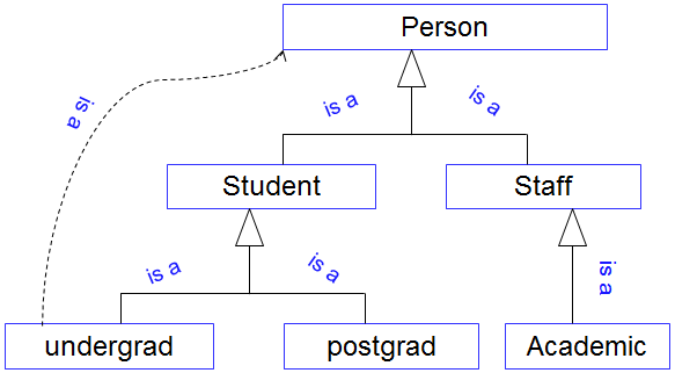
– We show relations among classes with different types of arrows.

| UML Relationship Symbols | | | |
|---|---|---|---|
| **Relationship** | **Symbol** | **Line Style** | **Arrow Tip** |
| inheritance | | Solid | Triangle |
| Interface implementation | | Dotted | Triangle |
| aggregation (“has-a”) | 1 | Solid | Diamond |
| dependency (“uses”) | | Dotted | Open |



# Inheritance Hierarchy

- A superclass itself can extend another class: e.g.
  - Customer extends Object
- The subclass implicitly inherits fields and methods in superclass including those the superclass inherited – and so on.




- Inheritance from superclasses of the direct superclass is called indirect inheritance.

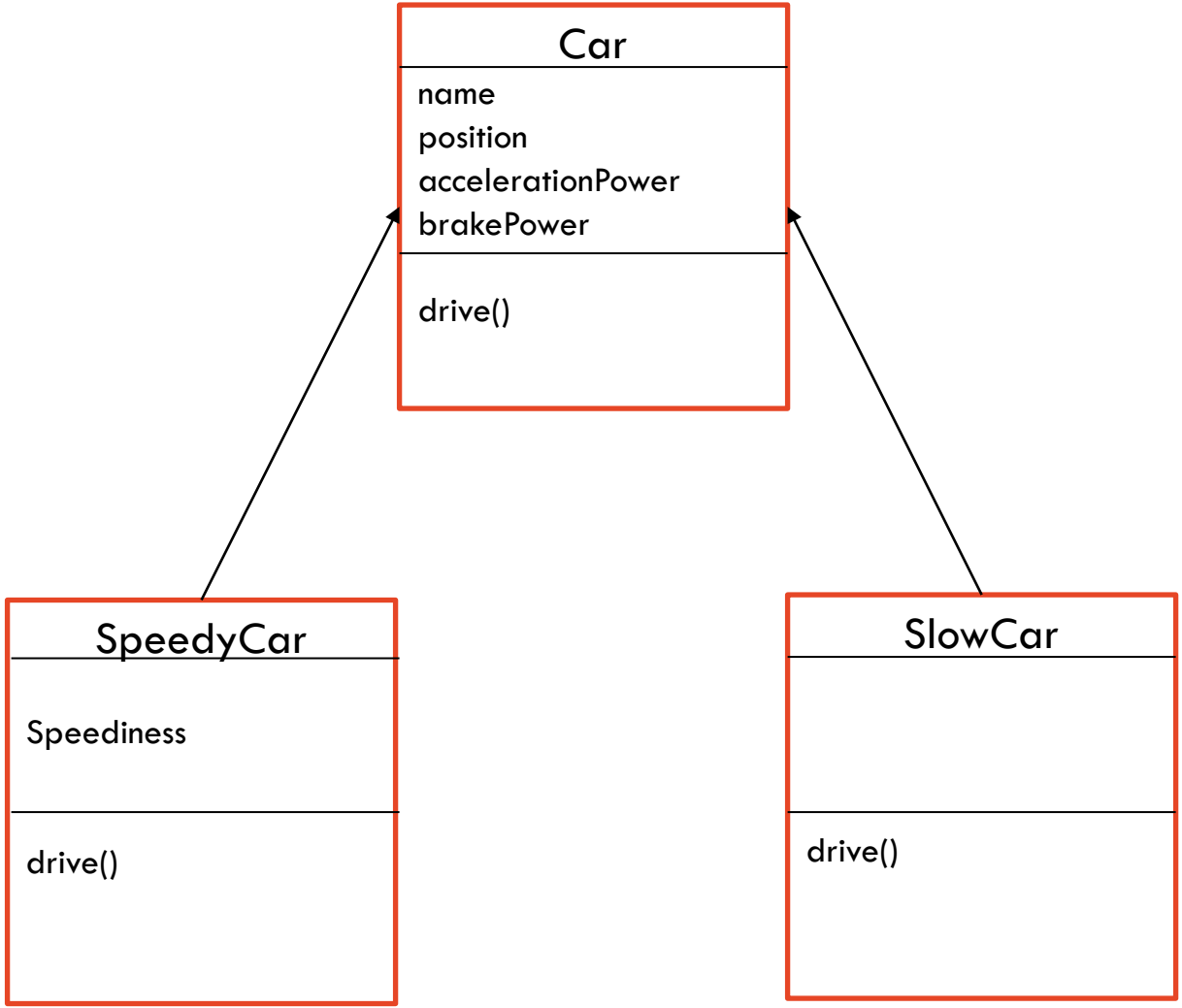
- A class can directly extend only one other class

- The classes form a tree based on extends relationship. This is the inheritance hierarchy

# Polymorphism via Inheritance

# Polymorphism

- Inheritance allows you to define a base class and derive subclasses from the base class
- Polymorphism allows you to make changes in the method implementation for the subclasses and have those changes achieved via the same method name as defined in the base class
- This allows
  - Dynamic binding (also known as late binding)


Car
name
position
accelerationPower
brakePower
drive()
SpeedyCar
Speediness
drive()
SlowCar
drive()

# Polymorphism

- In Java, type of a reference variable doesn't completely determine type of object to which it refers

```
Car c1 = new Car("Norman", 0, 1, 1);
Car c2 = new SpeedyCar("Zippy", 0, 1, 1, 1.3);
Car c3 = new SlowCar("Sleepy", 0, 1,1);
```

- Method calls are determined by type of actual object (using new and constructor), not type of object reference

```
c1.drive(1);
c2.drive(1);
c3.drive(1);
```

- This is an example of *polymorphism*: the ability to refer to objects of multiple types with varying behaviors

# Polymorphism

– In Java, type of a reference variable doesn't completely determine type of object to which it refers

```
Car c1 = new Car("Norman", 0, 1, 1);
c1.drive(1);
c1 = new SpeedyCar("Zippy", 0, 1, 1, 1.3);
c1.drive(1);
c1 = new SlowCar("Sleepy", 0, 1,1);
c1.drive(1);
```

This is an example of *polymorphism:* the ability to refer to objects of multiple types with varying behaviors

| Car object |
|---|
| Name="Norman"<br>position = 0;<br>accelerationPower = 1;<br>brakePower = 1 |
| drive() |

| SpeedyCar object |
|---|
| Name="Zippy"<br>position = 0;<br>accelerationPower = 1;<br>brakePower = 1<br>Speediness =1.3 |
| drive() |

| SlowCar object |
|---|
| Name="Sleepy"<br>position = 0;<br>accelerationPower = 1;<br>brakePower = 1 |
| drive() |

# Final Class

# Final Classes and Methods

- You can use the final keyword to prevent other programmers from creating subclasses or from overriding certain methods.
- For example, the String class in the standard Java library has been declared as:

  public final class String . . .

  - Thus nobody can extend the String class.
- You can also declare an individual method as final:

```
public class SecureAccount extends BankAccount {
. . .
        public final boolean checkPassword (String password)
        { . . . }
}
```

Nobody can override the ***checkPassword*** method with another method that simply returns *true*

# Final Classes and Methods

- ***final classes and methods***
  - a ***final*** method can't be overridden in a subclass
  - a ***final*** class can't be extended at all
  - ***final*** methods are safer
  - ***final*** methods improve runtime efficiency, as there is no need for late binding in this case.

# Questions?