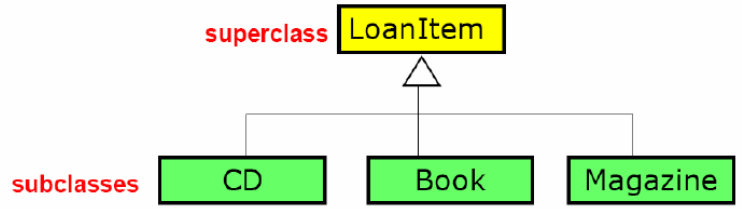
CSIT121

Object Oriented Design and Programming

Lesson 3

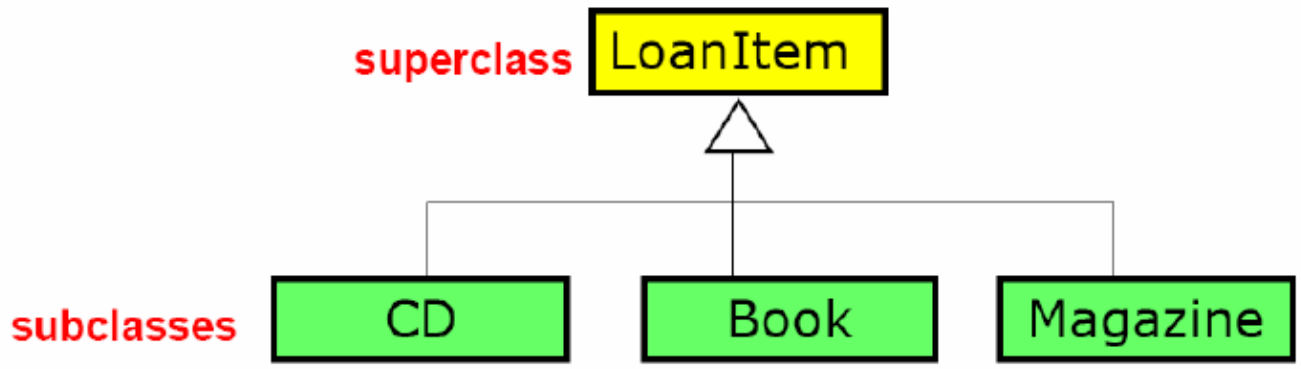
Inheritance

- Objects of different classes may share some common properties and behaviors.
- Example: For a Library system, there are many different types of items available for loan: CD, book, magazine, etc.
  - These items share some common properties and behaviors.
  - A superclass “LoanItem” can be defined.
  - This class defines the common properties and behaviors of all loan items.
    - CD, Book, Magazine classes inherits from LoanItem class.
    - CD, Book and Magazine classes are called subclasses.



Inheritance

- Subclasses inherit the attributes as well as methods of their superclass.
- However, each subclass has its own special attributes and methods as well that distinguish it from the other subclasses.
- Below is a tree-like structure that shows the relationships among classes.

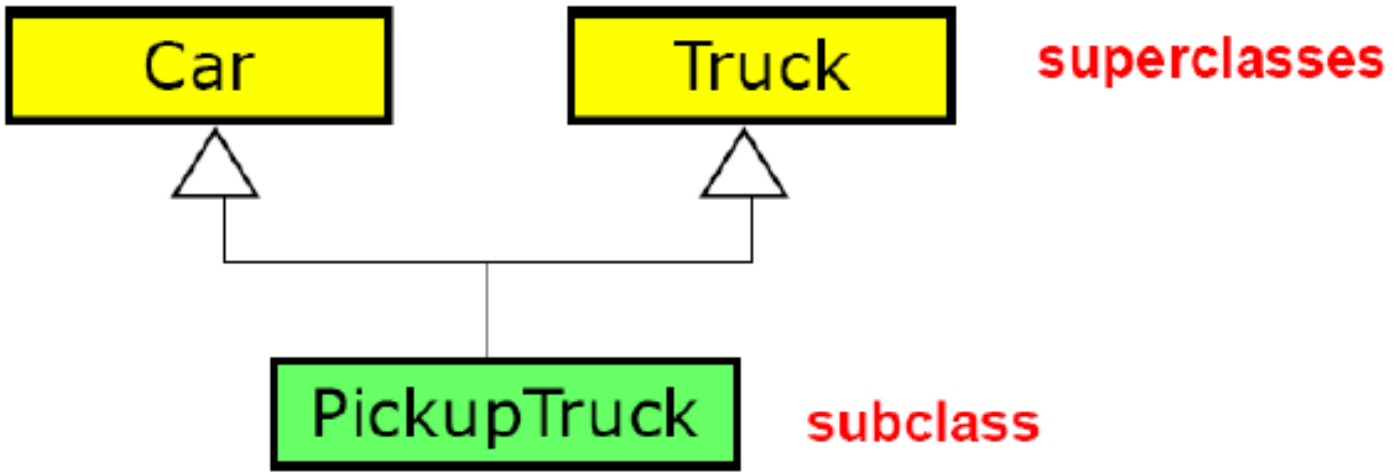


Inheritance

- Inheritance basically establishes a “is-a (type of)” relationship between superclass and its subclasses
  - CD is a type of LoanItem,
  - Book is a type of LoanItem,
  - Magazine is a type of LoanItem
- Each instance of the subclass is also an instance of the super class (The reverse is not true)
- For e.g: CD is a LoanItem but a LoanItem is not necessary a CD.

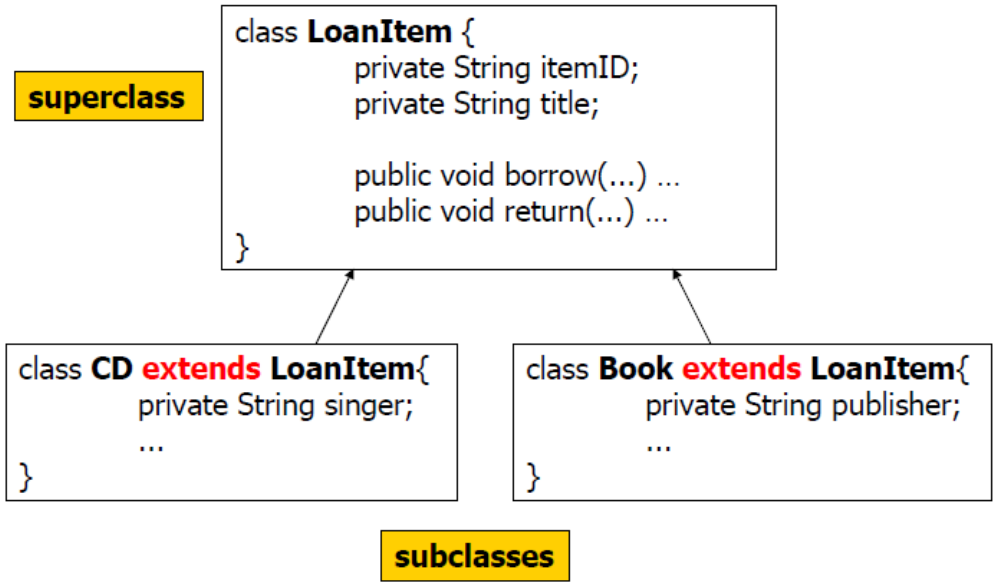
Multiple Inheritance

- A subclass may inherit from more than one superclasses
- PickupTruck inherits from 2 superclasses: Car and Truck
- Java only supports **single** inheritance.



Specifying Inheritance

- Use “extends” keyword
  - if nothing is specified, then “extends Object” is implicit
- Example
  - Book extends LoanItem – Book inherits the attributes and methods of LoanItem
- Subclass extends:
  - CD has its’ own attribute singer.
  - Book has its’ own attribute publisher.



Java Object class

- In Java, all classes are derived ultimately from the Object class (in java.lang package)
- If a class definition does not use “extends” clause to derive itself from another class, then the class is automatically derived from the Object class.

```
class LoanItem { //”extends Object” is implicitly stated
        private String itemID;
        private string title;

        public void borrow(...) …
        public void return(...) …
}
```

super Reference

- super is a reserved word in Java
- It refers to the corresponding superclass
- Using super reference, a subclass can access the members of its superclass.
- Commonly used to
  - access super class constructor.
  - access an overridden method in super class from subclasses (later)

```
class LoanItem {
        private String itemID;
        private string title;

        public LoanItem (String itemID, String title) {
                this.itemID= itemID;
                this.title= title;
        }
}
```

```
class CD extends LoanItem{
        private String singer;

public CD (String itemID, String title, String cdSinger) {
        super (itemID, title); // call the constructor
        singer = cdSinger;
        }
}
```

Constructor in subclasses

- Subclass does not inherit super constructor
- Subclass must call the super class constructor
- The calling of super class constructor may be implicit or explicit.

```java
public class SuperConstructorTest {

    public SuperConstructorTest() {
        System.out.println("In SuperConstructorTest");
    }
}
```

```java
public class SubClassConstructorTest extends SuperConstructorTest{

    public SubClassConstructorTest() {
        //will call super class default constructor implicitly
        System.out.println("In SubConstructorTest");
    }

    public static void main(String[] args) {
        SubClassConstructorTest t = new SubClassConstructorTest();
    }
}
```

```
<terminated> SubClassConstructorTest [Java Application] /Library
In SuperConstructorTest
In SubConstructorTest
```

Constructor in subclasses

- If the is no default constructor, need to call the super class constructor explicitly.

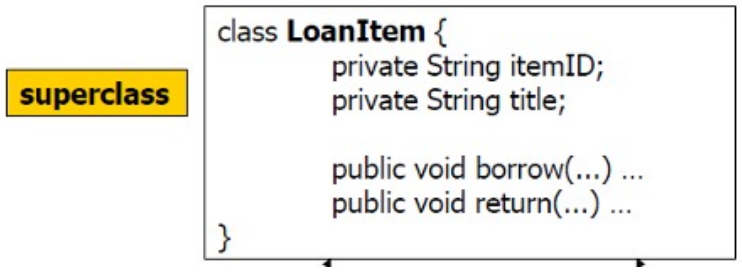
```java
public class SuperConstructorTest {

    public SuperConstructorTest(String msg) {
        System.out.println(msg);
    }
}
```

```java
public class SubClassConstructorTest extends SuperConstructorTest{

    public SubClassConstructorTest() {
        super("calling super class constructor");
        System.out.println("In SubConstructorTest");
    }

    public static void main(String[] args) {
        SubClassConstructorTest t = new SubClassConstructorTest();
    }
}
```

```
calling super class constructor
In SubConstructorTest
```

Private Members

- A superclass's private data members are not accessible by its subclasses.




```
class CD extends LoanItem{
        private Singer singer;
        public CD (String itemID, String title, Singer cdSinger)
        {
                super.itemID= itemID;
                super.title= title;
                singer = cdSinger;
        }
}
```

DIRECT access to private members is not allowed

Protected Members

- A superclass's protected data members are accessible by its subclasses and by other classes of the same package.

```
class LoanItem {
        protected String itemID;
        protected string title;
        ...
}
```

```
class CD extends LoanItem{
        private Singer singer;
        public CD (String itemID, String title, Singer cdSinger) {
                super.itemID= itemID;
                super.title= title;
                singer = cdSinger;
        }
}
```

DIRECT access to protected members is allowed

Method signature

- Methods in Java comprise
  - Access Modifier (e.g. public, private, protected)
  - Return Type
  - Method Name
  - Parameter List
  - Method Body

- The method signature comprises
  - Method Name
  - Parameter List
- Note: Return type is not part of method signature

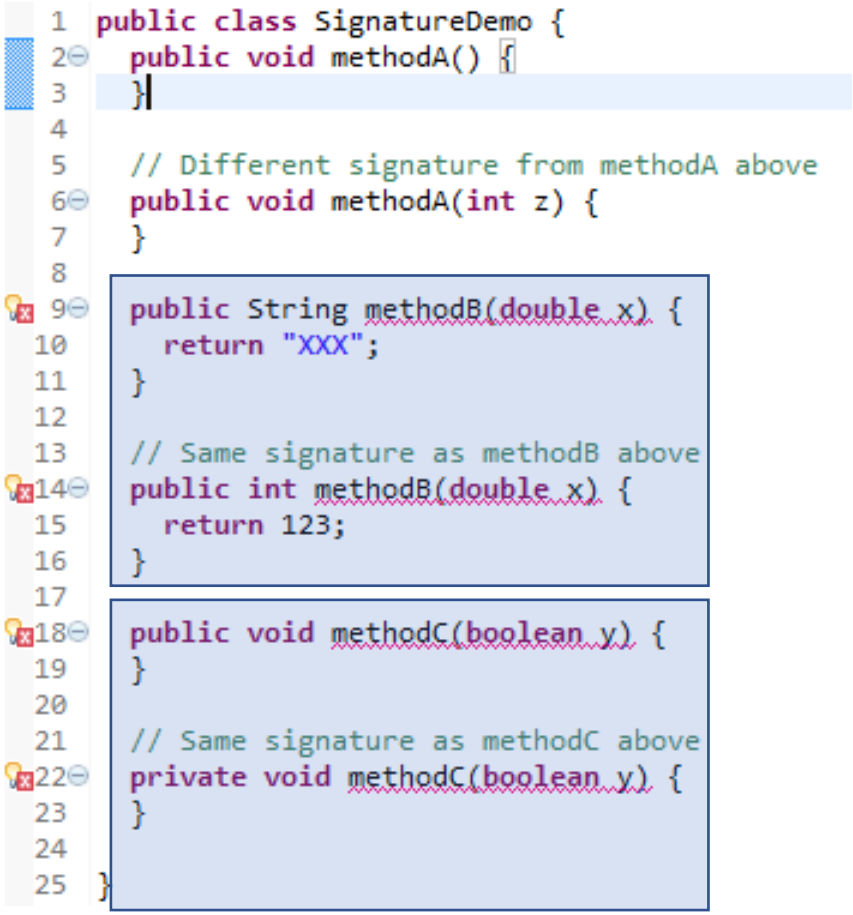
Method overload

- Method with the same name but different signature.
- Java will match the name and parameter list to determine which method to call.

```
public class SignatureDemo {

    public void methodA(){

    }

    public void methodA(int z){

    }

    public String methodB(double x){
        return "XXX";
    }

    public int methodB(int x){
        return 123;
    }

    public void methodC(boolean y){

    }

    public void methodC(char y){

    }
}
```

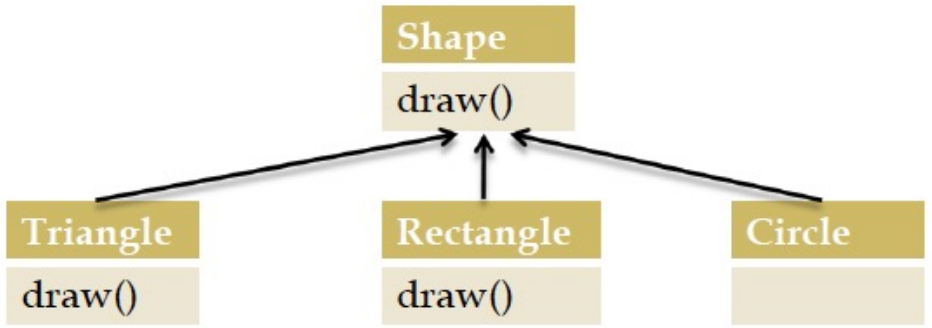
Method overload

- Compilation errors occur when two or more methods in a class have the same method signature.



Override method

- If a method from a super class is not “suitable” for the subclass, the subclass can implement a with the same method signature (i.e. same name and parameters) to override the super class method.
- In this example
  - Shape is the super class with draw() method.
  - Circle class inherit draw() from Shape.
  - Triangle and Rectangle override Shape.



Override method

```java
public class Shape {
    public void draw() {
        System.out.println("This is a shape");
    }
}
```

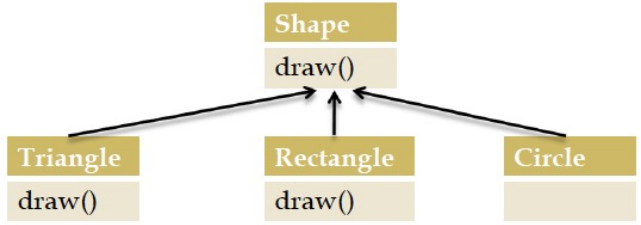
```java
public class Triangle extends Shape{
    public void draw() {
        System.out.println("This is a Triangle");
    }
}
```

```java
public class Rectangle extends Shape{
    public void draw() {
        super.draw();//call super class draw()
        System.out.println("This is a Rectangle");
    }
}
```

```java
public class Circle extends Shape{
    //inherit draw() from Shape
}
```

```java
public class ShapeTest {
    public static void main(String[] args) {
        Shape shape = new Shape();
        Triangle triangle = new Triangle();
        Rectangle rectangle = new Rectangle();
        Circle circle = new Circle();

        shape.draw();
        triangle.draw();
        rectangle.draw();
        circle.draw();
    }
}
```

```
Console
<terminated> ShapeTest [Java Application] /Library/Ja
This is a shape
This is a Triangle
This is a shape
This is a Rectangle
This is a shape
```



Abstract class

- An abstract class cannot be instantiated, and no object can be created from that abstract class.
- The purpose of abstract class is mainly for inheritance.

```
1
2  public abstract class AClass {
3
4      private int a;
5
6      public AClass(int a) {
7          this.a = a;
8      }
9
10     public String toString(){
11         return a+"";
12     }
13
14     public static void main(String args[]){
15
16         AClass a = new AClass(3);
17         AClass b = new AClass(6);
18         System.out.println(a);
19         System.out.println(b);
20     }
21 }
22
```

Final class

- A class that is “Final” cannot be inherited.
- That is the “family line” stops there.
- No more subclass below this class.

```
public final class AClass {

    private int a;

    public AClass(int a) {
        this.a = a;
    }

    public String toString(){
        return a+"";
    }

}
```

```
public class BClass extends AClass {


}
```

The type BClass cannot subclass the final class AClass

1 quick fix available:

Remove 'final' modifier of 'AClass'