Inheritance
Lecture 12

Waterford Institute of Technology

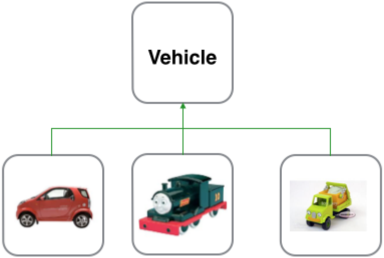
June 16, 2014

John Fitzgerald

Inheritance

Inheritance v Interfaces

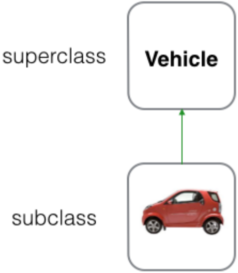
- Interfaces:
    - Unify behaviour
    - Cannot instantiate interface
- Inheritance:
    - Unify data & behaviour
- Vehicle has specific types
    - Common data
        - price, colour, speed
    - Common behaviour
        - start, move, stop



Inheritance

Terminology

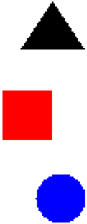
- Superclass
  - Class from which one inherits
  - Other names: *base*, *parent*
- Subclass
  - Class that inherits
  - Other names: *derived*, *extended*, *child*
- Vehicle
  - superclass of Car
- Car
  - subclass of Vehicle



Inheritance

Shapes

- Geometric shapes
  - Triangle, Circle, Rectangle
- Common data includes:
  - position, color
- Common behaviour includes:
  - moveTo, changeColor
- Class-specific behaviour
  - *draw()* implemented each subclass

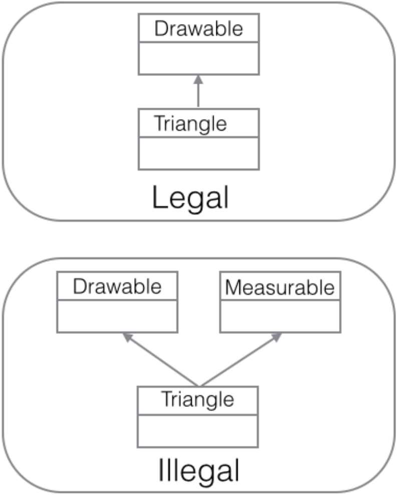


Inheritance

Inheritance v Interface

- Why not always use inheritance rather than interfaces?
  - Complexity: simpler to use interfaces
  - Class can inherit only from one class
  - Class can implement many interfaces

Class Diagrams

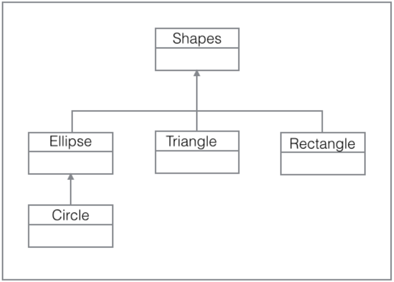


Inheritance

Levels of inheritance

- More levels more complexity
  - Difficult to know where fields and methods defined in deep hierarchies
  - Maximum one level used in this course

Class Hierarchy



Inheritance

Implement subclass

Subclass uses the ***extends*** keyword

The subclass may:

- directly use working methods in superclass
- override methods in superclass
- add new methods to subclass

The subclass

- may access the superclass fields
  - It should not redefine these
- may add new fields to subclass

```
public class Shapes
{
    ...
}
```

```
public class Rectangle extends Shapes
{
    ...
}
```

Inheritance

What to put in subclass

Essentially subclass has extra material not in superclass

- new methods required not already in superclass
- methods already in superclass that require changing
- additional instance variables

What not to include in subclass:

- methods already working in superclass
  - these are inherited from superclass
- superclass fields
  - these are also inherited from superclass

```
public class Shapes
{
    public void moveTo(int x, int y){...}
}
```

```
public class Rectangle extends Shapes
{
    public double area(){ return ...}
}
```

Inheritance

Inheriting & Overriding methods

Inherits

- *moveTo*

Overrides

- *makeVisible*

Added

- *area*

```
public class Shapes
{
    int xPos;

    public void moveTo(int x, int y){...}
    public void makeVisible(){...}
}
```

```
public class Rectangle extends Shapes
{
    public void makeVisible(){...}
    public double area(){...}
}
```

Inheritance

Subclass inherits & adds fields

Rectangle inherits superclass fields:

- *xPos*
- *yPos*

Rectangle adds new subclass fields:

- *xLen*
- *yLen*

```
public class Shapes
{
    int xPos;
    int yPos;
    ...

    public void moveTo(int x, int y){...}
    public void makeVisible(){...}
}
```

```
public class Rectangle extends Shapes
{
    int xLen;
    int yLen;

    public void makeVisible(){...}
    public double area(){...}
}
```

Inheritance

Instantiation: superclass

Shapes initializes its own fields

- *this.xPos = xPos;*
- Uses Rectangle constructor arguments

```
public class Shapes
{
    int xPos;
    int yPos;
    ...
    public Shapes(int xPos, int yPos)
    {
        this.xPos = xPos;
        this.yPos = yPos;
        ...
    }
}
```

Inheritance

Instantiation: subclass

Rectangle initializes its own fields

- *this.xLen = xLen;*

Rectangle initializes fields in superclass

- *super(xPos, yPos);*

```
public class Rectangle extends Shapes
{
    int xLen;
    int yLen;

    public Rectangle(int xLen, int yLen, int xPos, int yPos)
    {
        super(xPos, yPos);

        this.xLen = xLen;
        this.yLen = yLen;
    }
}
```

Java *interface*

Polymorphism

Term *polymorphism* already encountered in *Interfaces*

- Method invoked depends on invoking object
  - *triangleObj.makeVisible();*
  - *circleObj.makeVisible();*
- Allows building of expandable systems
- New types can be added without changing program logic
- Example
  - Instantiate new class, *Triangle extends Shapes*
  - Assign object to *Shapes* variable
  - Add new *Triangle* object to ArrayList *Shapes*
  - Repeat for other classes
  - Iterate list & invoke methods on referenced objects

Inheritance

Polymorphism

Example of polymorphism in action

- Create Circle, Rectangle & Triangle objects
- Add objects to ArrayList
- Iterate over array
- Invoke *makeVisible()* on each object in list

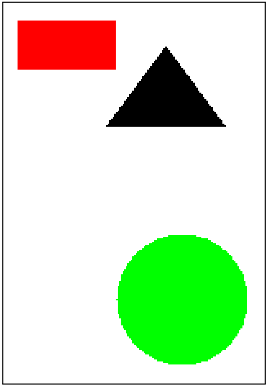
```
public static void main(String[] args) {
    ArrayList<Shapes> shapes = new ArrayList<>();
    shapes.add(new Triangle());
    shapes.add(new Circle());
    shapes.add(new Rectangle());

    for(Shapes shape : shapes) {
        shape.makeVisible();
    }
}
```

Inheritance

Polymorphism in action

- Three different *makeVisible* methods called:
  - Triangle's makeVisible
  - Circle's makeVisible
  - Rectangle's makeVisible

```
for(Shapes shape : shapes)
{
    shape.makeVisible();
}
```



Inheritance

Abstract class & method

In *Shapes* class method *makeVisible* not implemented

- *makeVisible* invokes *draw()*
- *draw* method different for each shape
- Therefore must implement in subclassses, not parent
- This necessitates declaration of *abstract makeVisible* in parent
- Also requires parent to be *abstract* class

```java
public abstact class Shapes
{
    //not implemented in Shapes
    //must be implemented in all derived classes
    abstract public void makeVisible();
}
```

Inheritance

package-private

Package: grouping of related types

- *shapes* package located in folder named **shapes**

Shapes: If no access level modifiers:

- *int xPos* is **package-private**
    - Inherited by all subclasses in package

```
package shapes;
public class Shapes
{
    int xPos;

}
```

```
package shapes;
public class Rectangle extends Shapes
{
    public moveHorizontal()
    {
        super.xPos += 1;
    }
}
```

Inheritance

Access control

Superclass private fields not visible in subclasses

- accessor required to read
- mutator required to modify

```
package shapes;
public class Shapes
{
    private int dimension;
    private void setDimension(int val)
    { ...}

}
```

```
package shapes;
public class Rectangle extends Shapes
{
    super.dimension = 1; //illegal
    super.setDimension(1); //illegal
}
```

Inheritance

Object class

All classes in Java descendent from **Object** class

- You may use or override some *Object* methods such as
  - *String toString()*
  - *int hashCode()*
  - *boolean equals(Object obj)*
- One class that it is not possible to override is:
  - *Class getClass()*

```
//Example using getClass: returns runtime class of this Object
package shapes;
public class TestShapes
{
  public static void main(String[] args) {
    Shapes shape = new Shapes();
    System.out.println(shape.getClass());
  }
}
//Output: class shapes.Shapes
```

Inheritance

Override Object.hashCode()

hashCode generates equivalent of unique ID

- Sample implementation:

```
package shapes;
public class Shapes {
  int xPosition;
  int yPosition;
  ...
  @Override
  public int hashCode() {
    Integer[] a = {xPosition, yPosition};
    return Arrays.hashCode(a);
  }
}
```

Inheritance

Override Object.equals()

equals determines if 2 objects equal

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (!(obj instanceof Shapes)) {
        return false;
    }
    Shapes other = (Shapes) obj;
    if (shapeFactor != other.shapeFactor) {
        return false;
    }
    return true;
}
```

Inheritance

Override Object.toString()

toString widely implemented

- Useful for debugging and logging
- Could use to translate object state to textual form
- No mandated style
- Eclipse default style used in sample code below

```
//Output: Shapes [shapeFactor=0]
package shapes;
public class Shapes {
    private int shapeFactor;
    @Override
    public String toString() {
        return "Shapes [shapeFactor=" + shapeFactor + "]";
    }
}
```

Referenced Material

1. Inheritance

http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[Accessed 2014-05-23]

2. Java Packages

http://docs.oracle.com/javase/tutorial/java/package/index.html

[Accessed 2014-05-24]

3. Object class

http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

[Accessed 2014-05-24]

Referenced Material (continued)

4. Polymorphism

`http://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html`

[Accessed 2014-06-16]