Advanced Object Orientation - Part I

CSC02A2

Outline

Outline

Inheritance

Inheritance

Definition

Reusing functionality from an existing class in a new class.

Key technique used for achieving reusability in OO programing.

Inheritance models an *is-a* relationship.

Java convention for naming refers to a superclass and a subclass. The functionality of the superclass is extended by the subclass. Java requires the use of the **`extend`** keyword to show inheritance.

All classes in Java descend from an ancestor class: **`java.lang.Object`**.

Using Inheritance

Subclasses contain more information and/or have more functionality than their superclass.

Not every *is-a* relationship should be modelled using inheritance, e.g. A **`Square`** is a *kind-of* **`Rectangle`** but a **`Square`** contains *less* information than a **`Rectangle`**.

Inheritance should not be used to reuse methods with similar functionality, e.g. a **`Person`** and a **`Rectangle`** both have height and width but a **`Person`** is not a **`Rectangle`**.

Multiple inheritance is not supported in **`Java`**.

Instance members with the **`protected`** visibility modifier can only be seen in the superclasses and its subclasses.

Subclasses cannot reduce the visibility of a superclass member.

The **super** keyword

Constructors in Java are not inherited but may be invoked in subclasses.

The **super** is used to refer to the superclass of the current class. It can be used to:

- Invoke a superclass constructor
- Invoke a superclass method

The invocation of the superclass constructor must be the first line in the subclass constructor. Every subclass has an *implicit* call to ***super()***.

The code below will not compile, why?

```java
class Fruit
{
  public Fruit(String s) { }
}
class Apple extends Fruit {}
```

Overriding methods and Casting

Overriding relies on two methods having the same return type as well the rest of the method signature the same. Static methods cannot be overridden.

Preventing overriding and subclassing:

```
// no subclasses for Mule allowed
public final class Mule { }
class SecuirtySystem
{
  // cannot override login
  public final boolean login() { }
}
```

Casting:

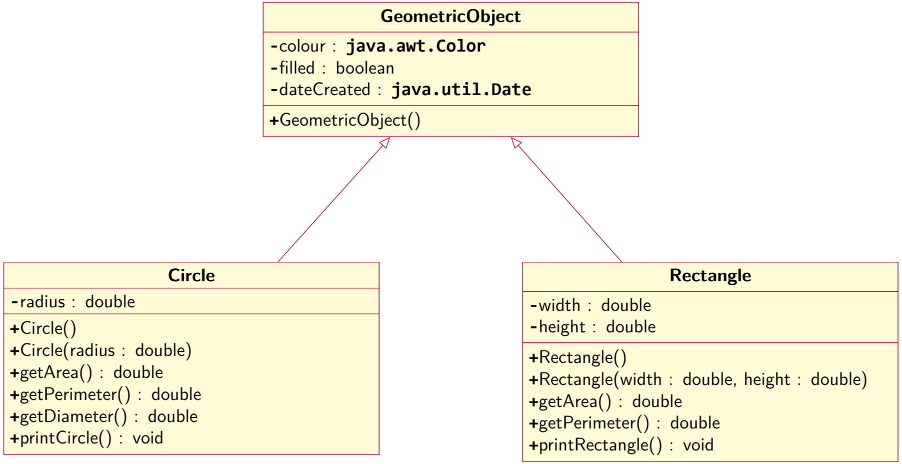
```
Object o = new Circle(); // Implicit casting
// This can result in a ClassCastException
Cirlce c = (Circle)o; // Explicit casting
// Checking before cast using instanceof
if(o instanceof Circle)
{
  Cirlce c = (Circle)o;
  c.printCircle();
}
```

java.lang.Object

The **Object** class defines a variety of useful methods:

- ***toString()*** provide a textual representation of an instance. By default provides the canonical name of the class and a hexadecimal representation of the hash of the instance.
- ***equals()*** test for instance equality as defined by the programmer. By default uses reference equality.

Inheritance Example I



Inheritance Example II

```java
import java.awt.Color;
import java.util.Date;
public class GeometricObject
{
    private Color colour = Color.WHITE;
    private boolean filled = false;
    private Date dateCreated = new Date();
    // Accessor methods omitted for brevity
    // Mutator methods omitted for brevity
    public String toString()
    {
        return "Created on " + dateCreated +
               "\nColour " + colour +
               "\nFilled: " + filled;
    }
}
```

Inheritance Example III

```java
public class Circle extends GeometricObject {
    private double radius;
    public Circle() {}
    public Circle(double newRadius) { setRadius(newRadius); }
    // Accessor methods omitted for brevity
    // Mutator methods omitted for brevity
    public double getArea()    {return Math.PI * Math.pow(radius, 2);}
    public double getDiameter(){ return 2 * radius;}
    public double getPerimiter(){ return getDiameter() * Math.PI;}
    public void printCircle()  { System.out.println(this);}
    public String toString()
    {
      return super.toString() + "\nradius: "
            + radius;
    }
}
```

Inheritance Example IV

```java
public class Rectangle extends GeometricObject
{
    private double width;
    private double height;

    public Rectangle() { }
    public Rectangle(double width, double height)
    {
       setWidth(width);
       setHeight(height);
    }
    // Accessor methods omitted for brevity
    // Mutator methods omitted for brevity
    public double getArea() { return width * height; }
    public double getPerimeter()
    {
      return 2 * (width + height);
    }
}
```