Abstract Classes, Overriding Methods, & Polymorphism

AP Computer Science

Abstract Classes

Abstract Classes

- Designed to be a parent to several related classes with shared implementations
- They can define methods and variables that children classes inherit
- Similar to interfaces, abstract classes are **never** instantiated
  - But they can have constructors!

```
public abstract class Shape {
   private int x, y;
      public Shape() { x = y = 0; }
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public int getX() { return x; }
   public int getY() { return y; }
}
```

Abstract Methods

- Abstract classes may include abstract methods (unimplemented), just like interfaces
  - The subclass MUST implement these methods, unless it too is an abstract class

```
public abstract class Shape {
   private int x, y;
      public Shape() { x = y = 0; }
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public int getX() { return x; }
   public int getY() { return y; }
   public abstract double getArea();
}
```

Abstract Classes

- Abstract class extending an abstract class

```
public abstract class Shape {
   private int x, y;
      public Shape() { x = y = 0; }
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public int getX() { return x; }
   public int getY() { return y; }
   public abstract double getArea();
}
```

```
public abstract class Shape3D extends Shape {
   private int z;
      public Shape3D() { z = 0; }
   public Shape3D(int xx, int yy, int zz)
      { super(xx,yy); z = zz; }
   public int getZ() { return z; }
   public abstract double getVolume();
}
```

Overriding Methods Example

- What is the output?

```
// main of another class
Circle cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getArea());
```

```
public abstract class Shape {
   private int x, y;
      public Shape() { x = y = 0; }
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public abstract double getArea();
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
      public Circle() { radius = 0; }
   public Circle(int x, int y, doubl
      { super(x, y); radius = r }
}
```

Output
Does not compile

- Circle has no getArea()

Overriding Methods Example

- What is the output?

```
// main of another class
Circle cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getArea());
```

```
public abstract class Shape {
   private int x, y;
      public Shape() { x = y = 0; }
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public abstract double getArea();
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
      public Circle() { radius = 0; }
   public Circle(int x, int y, double r)
      { super(x, y); radius = r }
   public double getArea() { return Math.PI * r * r; }
}
```

Output
$\pi(10.5)^2$

Overriding Methods

Overriding Methods

- This is when you replace the implementation of a method in the superclass.
- The overridden method must have the same method signature.
  - Method name, return type, and parameters.
- Can anyone think of a method we have overridden in the past?
  - `toString()`
  - `equals()`
  - `compareTo()`

Overriding Methods Example

- What is the output? Which getX() is called?

```
// main of another class
Circle cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getX());
```

Output
40

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public int getX() { return x; }
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
   public int getX() { return y; }  // very bad getX()
      /* hid other methods */ }
```

getX() in Circle

Denying Method Overriding

- How do you make a variable constant?
  - The `final` keyword signifies that a variable's value may not ever change.
- You can do a similar thing with methods, which tells Java that no subclass may override this method.

Overriding Methods Example

- What is the output?

```
// main of another class
Circle cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getX());
```

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public final int getX() { return x; }
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
   public int getX() { return y; }  // very bad getX()
   /* hid other methods */ }
```

Output

Does not compile

Cannot override final method

Polymorphism

Polymorphism

- Remember, we can refer to inheritance as an **is-a relationship**.
- Therefore, a variable can hold a reference to an object whose class is a descendant of the class of the variable
  - `Shape cir = new Circle(1, 2, 3.0);`
- You can call any method defined in **`Shape`**
- You can call any method defined in **`Shape`**, and overridden in **`Circle`**
- You can NOT the ones only defined in **`Circle`**
- This is an example of **polymorphism**, i.e. the ability of the **`Shape`** object to take on multiple forms

Polymorphism Example

- What is the output?

```
// main of another class
Circle cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getX());
```

getX() in Shape

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public final int getX() { return x; }
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
      /* hid other methods */ }
```

Output
50

Polymorphism Example

- What is different with this code? Will it work?

```
// main of another class
Shape cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getX());
```

getX() in Shape

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public final int getX() { return x; }
      /* hid other methods */ }
```

Output

50

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
      /* hid other methods */ }
```

Polymorphism Example

- What is different with this code? Will it work?

```
// main of another class
Shape cir1 = new Circle(50, 40, 10.5);
System.out.println(cir1.getRadius());
```

Output

Does not compile

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public final int getX() { return x; }
      /* hid other methods */ }
```

No getRadius() in Shape

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
   public double getRadius() { return radius; }
      /* hid other methods */ }
```

Polymorphism Example

- What is the output?

```
// main of another class
Shape cir = new Circle(50, 40, 10.5);
System.out.println(cir.getArea());
```

Output

$\pi(10.5)^2$

```
public abstract class Shape {
   private int x, y;
   public Shape(int xx, int yy) { x = xx; y = yy; }
   public abstract double getArea();
      /* hid other methods */ }
```

```
public class Circle extends Shape {
   private double radius;
   public Circle(int x, int y, double r)
      { super(x, y); radius = r; }
   public double getArea() { return Math.PI * r * r; }
}
```

getArea() in Circle

Polymorphism

- How do I know what methods I can call?
  - Consider the class hierarchy

- **`Shape`** can access it's own methods and **`Object`**'s
- **`Shape`** cannot access **`Circle`**'s methods

Object

```
public String toString()
public boolean equals(Object obj)
etc
```

Shape

```
public Shape(int x, int y)
public int getX()
public int getY()
public abstract double getArea()
```

Circle

```
public Circle(int x, int y, double r)
public int getRadius()
public double getArea()
```

Polymorphism

- What methods does **one** have access to?

```
Shape one = new Circle(50, 40, 10.5)
```

Object

```
public String toString()
public boolean equals(Object obj)
etc
```

Shape

```
public Shape(int x, int y)
public int getX()
public int getY()
public abstract double getArea()
```

Circle

```
public Circle(int x, int y, double r)
public int getRadius()
public double getArea()
```

Abstract Class Review

What is the output?

```
Animal animal = new Animal("Joe");
System.out.println(animal.speak());
```

```
public abstract class Animal {
    private String name;
    public Animal(String n) { name = n; }
    public abstract String speak();
}
```

You cannot instantiate an object of an abstract class

Output
Does not compile

Polymorphism

What is the output?

Output
Bark!

```
Animal animal = new Dog("Snoop");
System.out.println(animal.speak());
```

```
public abstract class Animal {
    private String name;
    public Animal(String n) { name = n; }
    public abstract String speak();
}
```

```
public class Dog extends Animal {
    public String speak() { return "Bark!"; }
}
```

Polymorphism

Consider the following subclasses

```
public abstract class Animal {
   // hid variable and constructor
   public abstract String speak(); }
```

```
public class Dog extends Animal {
   public String speak() { return "Bark!"; }
}
```

```
public class Cat extends Animal {
   public String speak() { return "Meow!"; }
}
```

```
public class Cow extends Animal {
   public String speak() { return "Moo!"; }
}
```

Polymorphism

What is the output?

```
ArrayList<Animal> list = new ArrayList<Animal>();
list.add(new Dog("Snoop"));
list.add(new Cat("Maru"));
list.add(new Cow("Bevo"));
for(int i = 0; i < list.size(); i++)
   System.out.println(list.get(i).speak());
```

Output
Bark!
Meow!
Moo!

- The list contains different implementations (subclasses) of **`Animal`**
  - ... but they all share the **`speak`** method.

References Allowed

Using the examples below, which would be allowed?

```
Animal animal;
Dog dog;
Cat cat;
Cow cow;
```

```
animal = new Cat();          // OK
dog = new Cat();             // Wrong
cat = new Cat();             // OK
cow = new Animal();          // Wrong
```