Overriding Methods & Polymorphism

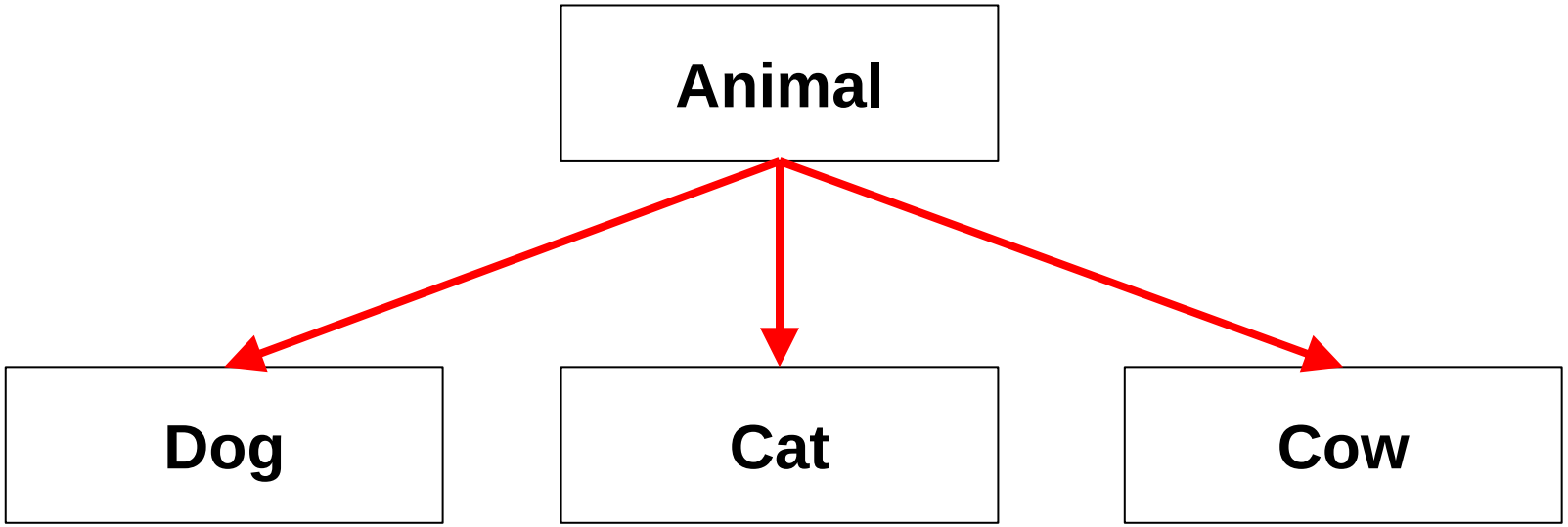
AP Computer Science

Overriding Methods

Overriding Methods

- This is when you replace the implementation of a method in the superclass
- For example, we might have an Animal superclass with Dog, Cat, and Cow subclasses
- 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()`

Animal Hierarchy



- Each class will have a speak method to allow the animal to speak
- Do all of these animals make the same sound?
  - No, you would want to give each one their own specific speak method

Overridden Methods

Which methods are overridden? Are any overloaded?

```
public class Animal {
   public String speak() { return "Speak!"; } }
```

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

Overridden

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

Overridden

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

Overridden

Overloaded

Overriding Methods Example

- What is the output? Which getArea() is called?
- Which method is overridden?

Output

$\pi(10.5)^2$

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

```
public class Shape {
   private int x, y;
   public Shape(int a, int b) { x = a; y = b; }
   public double getArea() { return 0; }}
```

```
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; }}
```

Note that getArea has the exact same method signature in both classes

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`**, but 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());
```

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

getX() in Shape

```
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());
```

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

getX() in Shape

```
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 the output?

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

```
public class Shape {
    private int x, y;
    public Shape(int xx, int yy) { x = xx; y = yy; }
    public int getX() { return x; }
    /* no getRadius() method */
}
```

getRadius() in Circle

```
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; }
}
```

Output
10.5

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());
```

```
public class Shape {
    private int x, y;
    public Shape(int xx, int yy) { x
    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 double getRadius() { return radius; }
    /* hid other methods */ }
```

Output

Does not compile

No getRadius() in Shape

Polymorphism Example

- What is the output?

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

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

getRadius() in Circle

```
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; }
}
```

Output
10.5

Polymorphism Example

- What is the output?

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

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

Output
$\pi(10.5)^2$

getArea() in Circle

Polymorphism

- How do I know what methods I can call?
  - Consider the class hierarchy - look at the object not the reference type

- **`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 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?

```
Object 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 double getArea()
```

Circle

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

Polymorphism

What is the output?

Output
Bark!

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

```
public class Animal {
    private String name;
    public Animal(String n) { name = n; }
    public String speak() { return "Speak!"; }
}
```

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

Polymorphism

Consider the following subclasses

```
public class Animal {
   // hid variable and constructor
   public String speak() { return "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;
```

A superclass can refer to a subclass, but a subclass cannot refer to a superclass

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