CS 213 – Software Methodology

Spring 2023

Sesh Venugopal

Feb 27

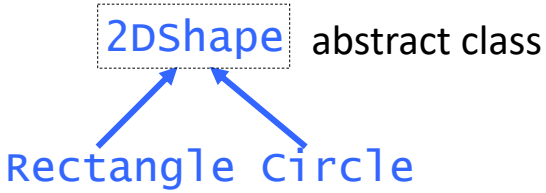
Abstract Classes

Abstract Classes – Introductory Examples

Rectangles and Circles have some common features:

- can be drawn on the 2D plane
- have a perimeter and an area
- it can be checked whether a point is inside or outside

The common features can be "abstracted" out into a superclass, say 2DShape.

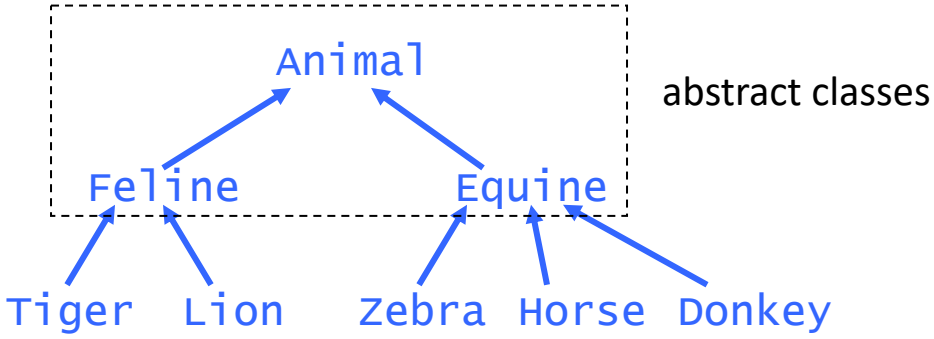


This is called GENERALIZATION: gathering properties that are common (general) to related classes into a superclass

But there is no actual 2DShape object:
only specific kinds of 2DShape objects.

So the generalized superclass is abstract

Abstract Classes – Introductory Examples



If you simulate an ecosystem, and populate it with specific kinds of animals, you cannot have an "Animal" or "Feline" or "Equine" object – you have to have "real" animals: Tigers or Lions, etc.

Abstract Class

When several classes have features,
i.e. traits (i.e. fields) and behaviors (i.e. methods),
in common, they can be **generalized**
into an abstract superclass.

Abstract Class – Java Definition

An abstract class MUST have keyword `abstract` in the class header

```
public abstract class 2DShape { … }
```

An abstract method has no implementation

```
public abstract class 2DShape {
   public abstract void draw();
   public abstract float area();
   ...
}
```

Abstract Class – Java Definition

An abstract class may have zero or more abstract methods.

```
public abstract class Device {
    protected String name;
    protected int widthPixelDensity;
    protected int heightPixelDensity;

    public String getName() {
        return name;
    }
    public int getWidthPixelDensity() {
        return widthPixelDensity;
    }
    public int getHeightPixelDensity() {
        return heightPixelDensity;
    }
}
```

This class has NO abstract methods

Abstract Class – Java Definition

An abstract class cannot be instantiated even if all methods have been implemented.

```
public abstract class Device {
   protected String name;
   protected int horizontalResolution;
   protected int verticalResolution;

   public String getName() {
      return name;
   }
   public int getHorizontalResolution() {
      return horizontalResolution;
   }
   public int getVerticalResolution() {
      return verticalResolution;
   }
}
```

```
Device device =
  new Device();
```

Abstract Class – Java Definition

Will this compile?

NO, because the class header does NOT have `abstract`, even though one of the methods is abstract

```
public class Vehicle {
    protected int numWheels;
    protected boolean hasMotor;

    public int getNumWheels() {
        return numWheels;
    }
    public boolean hasMotor() {
        return hasMotor;
    }
    public abstract int getWeight();
}
```

Abstract Class – Java Definition

An abstract class may implement constructors (if no constructor is implemented then the compiler will write in a default constructor)

```
public abstract class Device {
   protected String name;
   protected int horizontalResolution;
   protected int verticalResolution;

   public Device(String name,
                 int hres, int vres) {
      this.name = name;
      horizontalResolution = hres;
      verticalResolution = vres;
   }
   ...
}
```

So what's the point of having constructors if you can't create objects?

For use by "concrete" subclasses!

```
Device device = new Device("iPad Air",2048,1536);
```

Abstract Class – Java Definition

Constructors in an abstract class are for reuse by subclass constructors

```
public abstract class Device {
    ...
    public Device(String name,int hres, int vres) {
        ...
    }
    ...
}
public class iPad extends Device {
    String os;
    public iPad(String name,int hres, int vres, String os) {
        super(name, hres, vres);  // abstract superclass cons
        this.os = os;
    }
    ...
}
Device device = new iPad("iPad Air",2048,1536,"iOS 10.2.1");
```

Abstract Classes Example – Animal Hierarchy

```
public abstract class Animal {
    public void run() {
        System.out.println("run");
    }
}
```

```
public abstract class Feline
  extends Animal {
    public void purr() {
        System.out.println(”purr");
    }
}

public class Tiger extends Feline {
    public void purr() {
        System.out.print("Tiger: ");
        super.purr();
    }

    public void run() {
        System.out.print("Tiger: ");
        super.run();
    }
}
```

```
public abstract class Equine
  extends Animal {
    public void trot() {
        System.out.println("trot");
    }
}

public class Zebra extends Equine {
    public void trot() {
        System.out.print("Zebra: ");
        super.trot();
    }

    public void run() {
        System.out.print("Zebra: ");
        super.run();
    }
}
```

Class Polymorphism

```
public class Forest {
    public static void main(String[] args) {

        Animal[] animals = new Animal[5];
        animals[0] = new Tiger();
        animals[1] = new Lion();
        animals[2] = new Zebra();
        animals[3] = new Horse();
        animals[4] = new Donkey();
        for (int i=0; i < 5; i++) {
            animals[i].run();
        }

        Equine[] equines = new Equine[3];
        equines[0] = (Equine)animals[2];
        equines[1] = (Equine)animals[3];
        equines[2] = (Equine)animals[4];
        for (int i=0; i < 3; i++) {
            equines[i].trot();
        }

    }
}
```

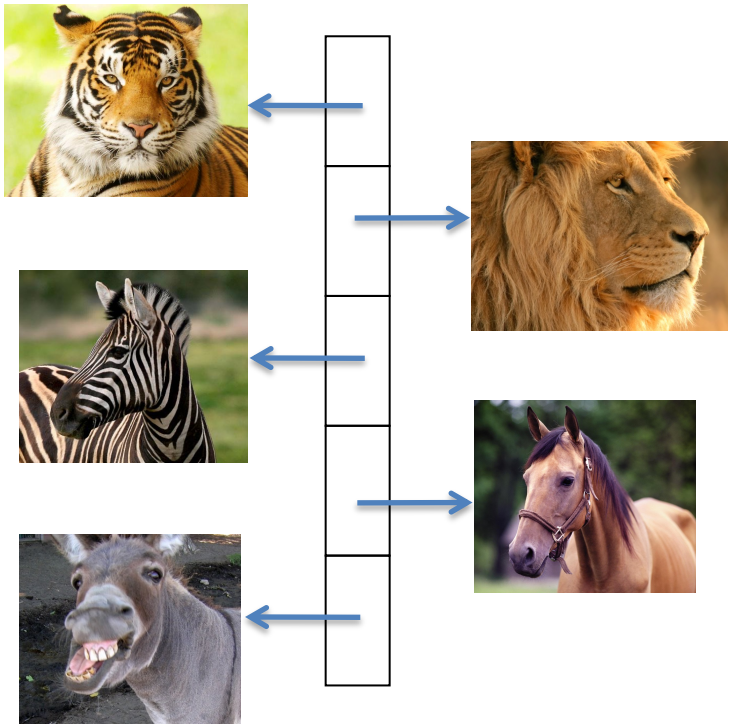
Static/ compile-time type is `Animal`

Dynamic/ run-time types are different

Polymorphism

Polymorphism

```
> java Forest
Tiger: run
Lion: run
Zebra: run
Horse: run
Donkey: run
Zebra: trot
Horse: trot
Donkey: trot
>
```



Java FX Example: Application Class

- `javafx.application.Application` is an abstract class with several non-abstract static and instance methods, and a single abstract method, namely `start`
  - `public abstract void start(Stage stage) throws Exception`

Non-GUI Abstract Classes

- Example: `java.util.Dictionary`
  - A dictionary is a data structure that allows insert, search, and delete – all methods of `Dictionary` are abstract
  - Any search structure (e.g. hash table) can implement `Dictionary: java.util.Hashtable` is a concrete subclass of Dictionary
- The `Dictionary` class is now obsolete, replaced by `java.util.Map` interface – why?
- The `java.util.HashMap` class implements the `Map` interface, but it also extends the `java.util.AbstractMap` abstract class – why?

Non-GUI Abstract Classes

- Example: `java.util.Calendar`
  - Provides methods to convert between specific instant in time and calendar attributes year, day, month, etc.
  - `java.util.GregorianCalendar` is a concrete subclass of `Calendar`
  - Static method `Calendar.getInstance()` returns calendar instance for current time with default locale (US – Gregorian calendar)

Abstract Class versus Interface?

(Popular Interview Question!!)