Abstract classes and interfaces

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 4

Grouping together classes

- Sometimes we collect together classes under a common heading

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes
- Create a class `Shape` so that `Circle`, `Square` and `Rectangle` extend `Shape`

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes
- Create a class `Shape` so that `Circle`, `Square` and `Rectangle` extend `Shape`
- We want to force every `Shape` to define a function
  `public double perimeter()`

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes
- Create a class `Shape` so that `Circle`, `Square` and `Rectangle` extend `Shape`
- We want to force every `Shape` to define a function
  `public double perimeter()`
- Could define a function in `Shape` that returns an absurd value
  `public double perimeter() { return(-1.0); }`

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes
- Create a class `Shape` so that `Circle`, `Square` and `Rectangle` extend `Shape`
- We want to force every `Shape` to define a function
  `public double perimeter()`
- Could define a function in `Shape` that returns an absurd value
  `public double perimeter() { return(-1.0); }`
- Rely on the subclass to redefine this function

Grouping together classes

- Sometimes we collect together classes under a common heading
- Classes `Circle`, `Square` and `Rectangle` are all shapes
- Create a class `Shape` so that `Circle`, `Square` and `Rectangle` extend `Shape`
- We want to force every `Shape` to define a function
  `public double perimeter()`
- Could define a function in `Shape` that returns an absurd value
  `public double perimeter() { return(-1.0); }`
- Rely on the subclass to redefine this function
- What if this doesn't happen?
  - Should not depend on programmer discipline

Abstract classes

- A better solution
  - Provide an abstract definition in `Shape`

```
public abstract double perimeter();
```

Abstract classes

- A better solution
  - Provide an abstract definition in `Shape`

```
public abstract double perimeter();
```

- Forces subclasses to provide a concrete implementation

Abstract classes

- A better solution
  - Provide an abstract definition in `Shape`

```
public abstract double perimeter();
```

- Forces subclasses to provide a concrete implementation
- Cannot create objects from a class that has abstract functions

Abstract classes

- A better solution
  - Provide an abstract definition in `Shape`

    ```
    public abstract double perimeter();
    ```
- Forces subclasses to provide a concrete implementation
- Cannot create objects from a class that has abstract functions
- `Shape` must itself be declared to be `abstract`

```
public abstract class Shape{
  ...
  public abstract double perimeter();
  ...
}
```

Abstract classes . . .

- Can still declare variables whose type is an abstract class

Abstract classes . . .

- Can still declare variables whose type is an abstract class

```
Shape shapearr[] = new Shape[3];
int sizearr[] = new int[3];

shapearr[0] = new Circle(...);
shapearr[1] = new Square(...);
shapearr[2] = new Rectangle(...);

for (i = 0; i < 2; i++){
  sizearr[i] = shapearr[i].perimeter();
     // each shapearr[i] calls the appropriate method
  ...
}
```

Generic functions

- Use abstract classes to specify generic properties

```
public abstract class Comparable{
  public abstract int cmp(Comparable s);
    // return -1 if this < s,
    //         0 if this == 0,
    //        +1 if this > s
}
```

Generic functions

- Use abstract classes to specify generic properties

```
public abstract class Comparable{
  public abstract int cmp(Comparable s);
    // return -1 if this < s,
    //         0 if this == 0,
    //        +1 if this > s
}
```

- Now we can sort any array of objects that extend `Comparable`

```
public class SortFunctions{
  public static void quicksort(Comparable[] a){
    ...
    // Usual code for quicksort, except that
    // to compare a[i] and a[j] we use a[i].cmp(a[j])
  }
}
```

Generic functions . . .

```
public class SortFunctions{
  public static void quicksort(Comparable[] a){
    ...
  }
}
```

Generic functions . . .

```
public class SortFunctions{
  public static void quicksort(Comparable[] a){
    ...
  }
}
```

- To use this definition of `quicksort`, we write

```
public class Myclass extends Comparable{
  private double size;   // quantity used for comparison

  public int cmp(Comparable s){
    if (s instanceof Myclass){
      // compare this.size and ((Myclass) s).size
      // Note the cast to access s.size
    }
  }
}
```

Mutiple inheritance

- Can we sort `Circle` objects using the generic functions in `SortFunctions`?
  - `Circle` already extends `Shape`
  - Java does not allow `Circle` to also extend `Comparable`!

Mutiple inheritance

- Can we sort `Circle` objects using the generic functions in `SortFunctions`?
  - `Circle` already extends `Shape`
  - Java does not allow `Circle` to also extend `Comparable`!
- An interface is an abstract class with no concrete components

```
public interface Comparable{
  public abstract int cmp(Comparable s);
}
```

Mutiple inheritance

- Can we sort `Circle` objects using the generic functions in `SortFunctions`?
  - `Circle` already extends `Shape`
  - Java does not allow `Circle` to also extend `Comparable`!
- An interface is an abstract class with no concrete components

```
public interface Comparable{
  public abstract int cmp(Comparable s);
}
```

- A class that extends an interface is said to implement it:

```
public class Circle extends Shape implements Comparable{
  public double perimeter(){...}
  public int cmp(Comparable s){...}
     ...
}
```

Mutiple inheritance

- Can we sort `Circle` objects using the generic functions in `SortFunctions`?
  - `Circle` already extends `Shape`
  - Java does not allow `Circle` to also extend `Comparable`!
- An interface is an abstract class with no concrete components

```
public interface Comparable{
  public abstract int cmp(Comparable s);
}
```

- A class that extends an interface is said to implement it:

```
public class Circle extends Shape implements Comparable{
  public double perimeter(){...}
  public int cmp(Comparable s){...}
     ...
}
```

- Can extend only one class, but can implement multiple interfaces

Summary

- We can use the class hierarchy to group together related classes
- An abstract method in a parent class forces each subclass to implement it in a sensible manner
- Any class with an abtract method is itself abstract
  - Cannot create objects corresponding to an abstract class
  - However, we can define variables whose type is an abstract class
- Abstract classes can also describe capabilities, allowing for generic functions
- An interface is an abstract class with no concrete components
  - A class to extend only one parent class, but it can implement any number of interfaces