# OOP with Java

## 23. Abstract Classes

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

## Outline

website

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`
- There is no meaningful way to implement this method in class `Shape`

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`
- There is no meaningful way to implement this method in class `Shape`
- But if we have a class `Rectangle extends Shape`, this class can implement it in meaningfully

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`
- There is no meaningful way to implement this method in class `Shape`
- But if we have a class `Rectangle extends Shape`, this class can implement it in meaningfully
- The default solution would be to define the method as `void print(){ }` in `Shape` so that it does nothing

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`
- There is no meaningful way to implement this method in class `Shape`
- But if we have a class `Rectangle extends Shape`, this class can implement it in meaningfully
- The default solution would be to define the method as `void print(){ }` in `Shape` so that it does nothing
- But this is not always possible (say, if a return value needs to be generated) and also does not force subclasses of `Shape` to override the method

## Introduction

- Sometimes, we create a base class with a method which can only meaningfully be implemented in a subclass
- Imagine a base class `Shape` for geometric shapes with a method `void print();` which should print an outline of the shape to `stdout`
- There is no meaningful way to implement this method in class `Shape`
- But if we have a class `Rectangle extends Shape`, this class can implement it in meaningfully
- The default solution would be to define the method as `void print(){ }` in `Shape` so that it does nothing
- But this is not always possible (say, if a return value needs to be generated) and also does not force subclasses of `Shape` to override the method
- How can we *a)* define `print` properly in class `Shape` and *b)* force all subclasses to implement this method?

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented
- An `abstract` class cannot be instantiated, since it has unimplemented methods

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented
- An `abstract` class cannot be instantiated, since it has unimplemented methods
- An `abstract` class can have a subclass `B` which is not `abstract`

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented
- An `abstract` class cannot be instantiated, since it has unimplemented methods
- An `abstract` class can have a subclass `B` which is not `abstract`
- Such a subclass `B` must override and implement all inherited `abstract` methods

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented
- An `abstract` class cannot be instantiated, since it has unimplemented methods
- An `abstract` class can have a subclass `B` which is not `abstract`
- Such a subclass `B` must override and implement all inherited `abstract` methods
- Since it is a subclass, you can store an instance of `B` in a variable of type `A`

## Abstract Classes

- An `abstract` class `A` is a class which can have `abstract` methods
- An `abstract` method is a method which has no method body, i.e., is only specified but not implemented
- An `abstract` class cannot be instantiated, since it has unimplemented methods
- An `abstract` class can have a subclass `B` which is not `abstract`
- Such a subclass `B` must override and implement all inherited `abstract` methods
- Since it is a subclass, you can store an instance of `B` in a variable of type `A`
- You can, of course, call all the methods of such a variable, even the `abstract` ones, because there cannot be any instance with an `abstract` method

## Example for an `abstract` class

Listing: Example for an abstract class with an abstract method

```java
/** the abstract class Shape */
public abstract class Shape {
  /** the print method is not yet implemented */
  public abstract void print();
}
```

## Subclass overriding `abstract` method

Listing: Subclass overriding abstract method

```java
/** the non-abstract class Rectangle extends the abstract class Shape */
public class Rectangle extends Shape {
  /** the width */
  private int width;
  /** the height */
  private int height;

  /** create the rectangle */
  public Rectangle(final int w, final int h) {
    this.width = w; this.height = h;
  }

  /** print the rectangle */
  public void print() {
    for(int i = 0; i < this.height; i++) {
      for(int j = 0; j < this.width; j++) {
        System.out.print('#');
      }
      System.out.println();
    }
  }
  /** The main routine
   *  @param args  we ignore this parameter */
  public static void main(String[] args) {
    Shape rectangle = new Rectangle(10, 5); // We can store Rectangles in Shape variables
    rectangle.print();                      // and invoke the print method
  }
}
```

## Another subclass overriding `abstract` method

Listing: Another subclass overriding abstract method

```java
/** the non-abstract class Circle extends the abstract class Shape */
public class Circle extends Shape {
  /** the radius */
  private int radius;

  /** create the circle*/
  public Circle(final int r) {
    this.radius = r;
  }

  /** print the circle */
  public void print() {
    int range = 2 * this.radius;
    for(int i = 0; i < range; i++) {
      for(int j = 0; j < range; j++) {
        System.out.print(
            ((int)(0.5d + Math.hypot(i-this.radius, j-this.radius))) < this.radius
            ? '#' : '␣');
      }
      System.out.println();
    }
  }
  /** The main routine
   *  @param args  we ignore this parameter */
  public static void main(String[] args) {
    Shape circle = new Circle(11); // We can store Circles in Shape variables
    circle.print();                // and invoke the print method
  }
}
```

## Summary

- We have learned about `abstract` classes
- `abstract` class cannot be instantiated, only subclassed (extended)
- `abstract` classes can have `abstract` methods, which are methods without implementation
- Their non-`abstract` subclass then need to override and implement these methods
- This is a way for us to define base classes which have methods that cannot be implemented for these base classes and force any user subclassing our class to implement them

# 谢谢

# Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, "Der Wanderer über dem Nebelmeer", 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog