Abstract Classes and Interfaces

ABSTRACT CLASSES AND INTERFACES

- **Objectives**
  - Improve design by using abstract classes
- **Contents**
  - The problem if we have no abstract classes
  - Abstract classes with abstract members
  - Polymorphism
- **Hands-on labs**
  - Implementing interfaces

WHAT ARE ABSTRACT CLASSES

- **You would not create and draw a shape**
  - You would draw a derived type of Shape (rectangle, circle…)
- **You would never open an account**
  - You would open a type of Account (savings, current…)
- **Some classes are not meant to be instantiated**
  - These are marked as abstract

- **abstract classes follow these rules:**
  - They can hold **fields, methods, constructors**
  - May have zero or more **methods** marked as **abstract**
  - Cannot be instantiated
  - Are a base for inheritance
    - **e.g.** what all shapes have in common (x, y, w, h, colour…)

Problem with a concrete base class

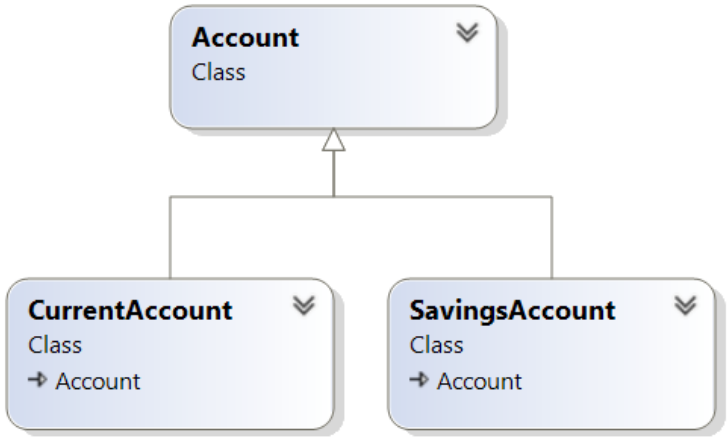
```
Account[] accounts = { new CurrentAccount(),
                       new SavingsAccount() };

for (Account acc : accounts) {
     acc.withdraw(50);
}
```

```
class Account {
   double balance;
}

class CurrentAccount extends Account {
   public void withdraw(int amt) {
        print("CurrentAccount withdraw");
   }
}

class SavingsAccount extends Account {
   public void withdraw(int amt) {
         print("SavingAccount withdraw");
   }
}
```



Will this code compile?

Problem with a concrete base class…

```
Account[] accounts = { new CurrentAccount(),
                       new SavingsAccount() };

for (Account acc : accounts) {
        acc.withdraw(50);
}
```

```
class Account {
   double balance;
   public void withdraw(int amt) { }
}

class CurrentAccount extends Account {
   public void withdraw(int amt) {
         print('CurrentAccount withdraw');
   }
}

class SavingsAccount extends Account {
   public void withdraw(int amt) {
          print('SavingAccount withdraw');
   }
}
```

Add a withdraw() method

What is the problem now?

You can override but don't have to.

But, you can create an instance of Account

Using an abstract class – Problem solved!

```
Account[] accounts = { new CurrentAccount(),
                       new SavingsAccount() };

for (Account acc : accounts) {
    acc.withdraw(50);
}
```

Cannot create an instance of `abstract` Account

```
abstract class Account {
   double balance;
   public abstract void withdraw(int amt);
}

class CurrentAccount extends Account {
   public void withdraw(int amt) {
         balance -= amt + 1;
   }
}

class SavingsAccount extends Account {
   public void withdraw(int amt) {
         balance -= amt;
   }
}
```

No code

`abstract` methods live in `abstract` classes.

Have no code, as they cannot meaningfully be implemented

`abstract` methods must be overridden in every derived class

Polymorphism with abstract classes

```
Shape[] shapes = { new Rectangle(), new Circle() };
    for (Shape shape : shapes) {
        draw(shape);
    }

void draw(Shape shape) {
    shape.draw();
}
```

```
abstract class Shape {
    public abstract void draw();
}
```

```
class Rectangle extends Shape {
    public void draw() {
        // code to draw a rectangle
    }
}
class Circle extends Shape {
    public void draw() {
        // code to draw a circle
    }
}
```

Shape.draw()
is handled by the derived type.
In future could handle as yet unwritten derived types

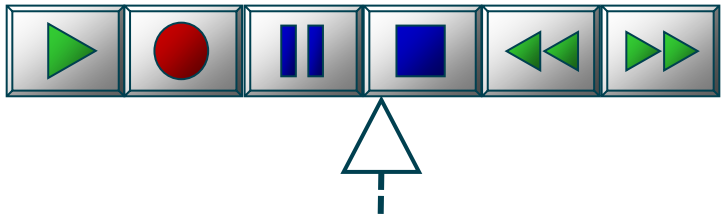
POWERING POTENTIAL

Interfaces

Interfaces

An interface is similar to a fully abstract class

- All of its members are abstract, no implementation code
- Can have no instance fields, although constants allowed
- Clearly cannot be instantiated
- Useful if you want to define a role that could be played by class



```
class MediaPlayer implements Playable  {
}
```

```
interface Playable {
    void play();
    void stop();
    void fastForward(int x);
    void fastRewinde();
    void record();
    void pause();
}
```

Defining an interface type

- **Interface defined using keyword `interface`**
  - Often end in '...able' no specific naming convention otherwise

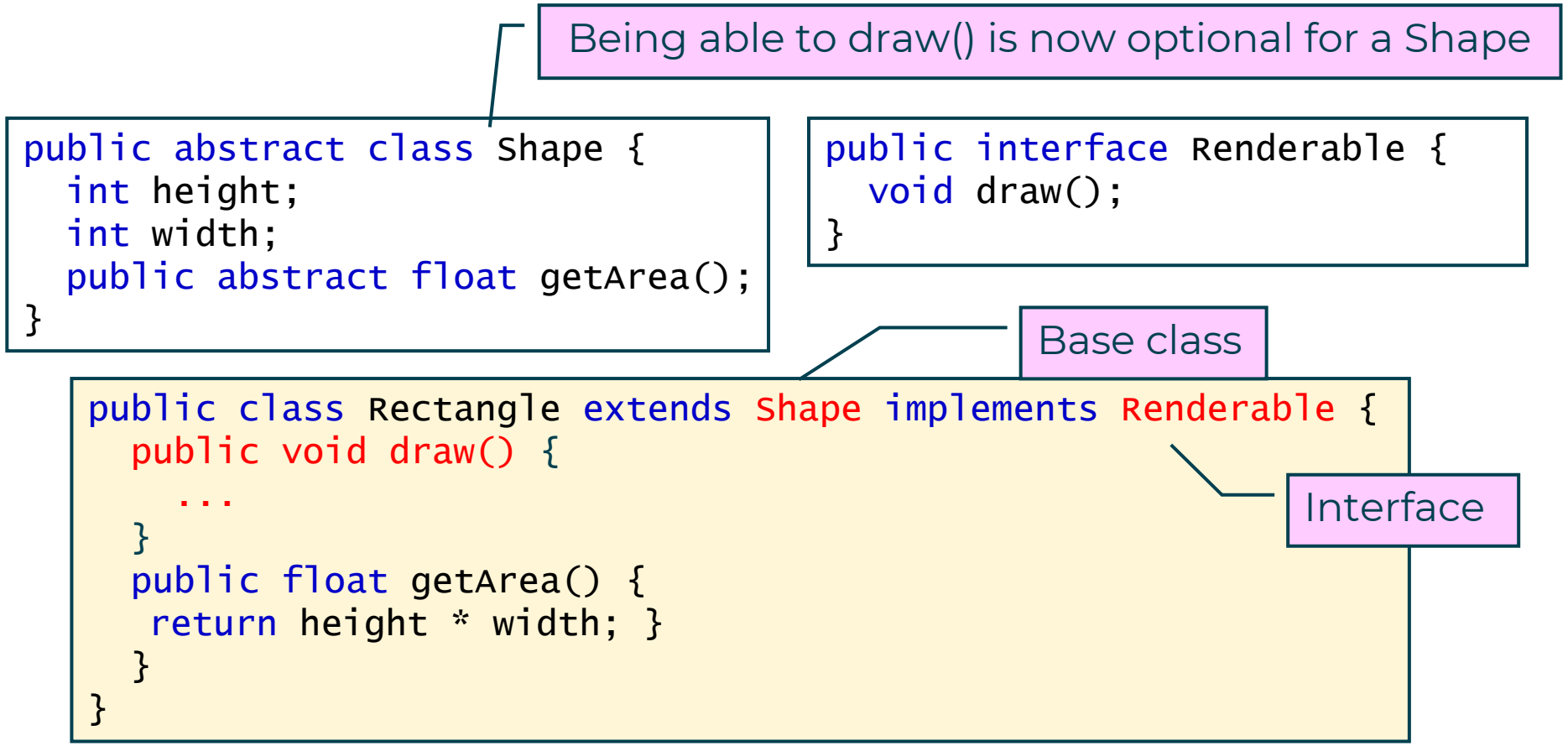
- **Interface members**
  - All methods are implicitly `public , abstract and non-static`

```
public interface Renderable {
  void draw();
}
```

Implementing an interface

List interfaces after the base class (if any) via keyword implements

- All members must be implemented



Polymorphism again

An interface defines a new type, just like a class

- If method has parameter of an interface type, it can be passed a reference to an object of any class that implements the interface
- Can also have collections of objects that implement a specific interface

```
public class Canvas {
  private Shape[] shapes;
  private void renderRenderables() {
    for( Shape s : shapes ) {
      if (s instancof Renderable)
        ((Renderable)s).draw();
    }
  }
}
```

Stores ‘shape’ refs

‘Dynamic’ cast produces an Renderable reference

```
public class Canvas {
  private void processRenderable(Renderable ir) {
    ir.draw();
  }
}
```

Multiple interfaces

A class can implement multiple interfaces

- You do not *inherit* from an interface, you *implement* it

```
public interface Comparable {
  int compareTo( Object o1 );
}
```

```
public interface Renderable {
  void draw();
}
```

```
public class Rectangle extends Shape
                       implements Comparable, Renderable {
  private int height;
  private int width;

  public void draw() {
    ...
  }
  public int compareTo( Object o ) {
    Rectangle r = (Rectangle)o;
    return ...
  }
}
```

Review – Why use abstract classes?

- **You do not want anyone to create an instance**
  - but you can always make a constructor private in any concrete class
- **Can Abstract methods have code? No!** Must be implemented by a sub class.
- **They set the pattern for extended classes**
- **They can contain state and code**
  - fields, getters and setters, other methods
- **Form the basis for any extended class**
  - **Is also a contract.** You must implement this method
- Can an abstract class **extend another abstract** class? **Yes**.
- Can an abstract class **implement an interface**? **Yes**

- **How many abstract classes can you extend from**? **1** and Only One and no more!

Review – Why use interfaces?

- They are **pure abstract** classes
- **Can they contain state** (variables)? **No!**
- Can they contain **concrete methods** (with code)? **No!**
- Interfaces such as drawable, comparable, closable,… **what do these shows**?
  - the **capabilities of an object** that implements an interface
  - In other words **What actions** can they perform
- Can an **abstract class implement an interface**? **Yes**.
- Can an **interface implement another interface**? **Yes**
- Can an interface have **getters and setters**? **Yes.** These are just methods

  - **How many interfaces can you implement** in a single class? many and **as many as you like**