Strategy pattern

Lecture 11

Inheritance syntax reminder

- Class Art
- Drawing extends Art
- Cartoon extends Drawing

- The object of class cartoon is created from the base class outwards

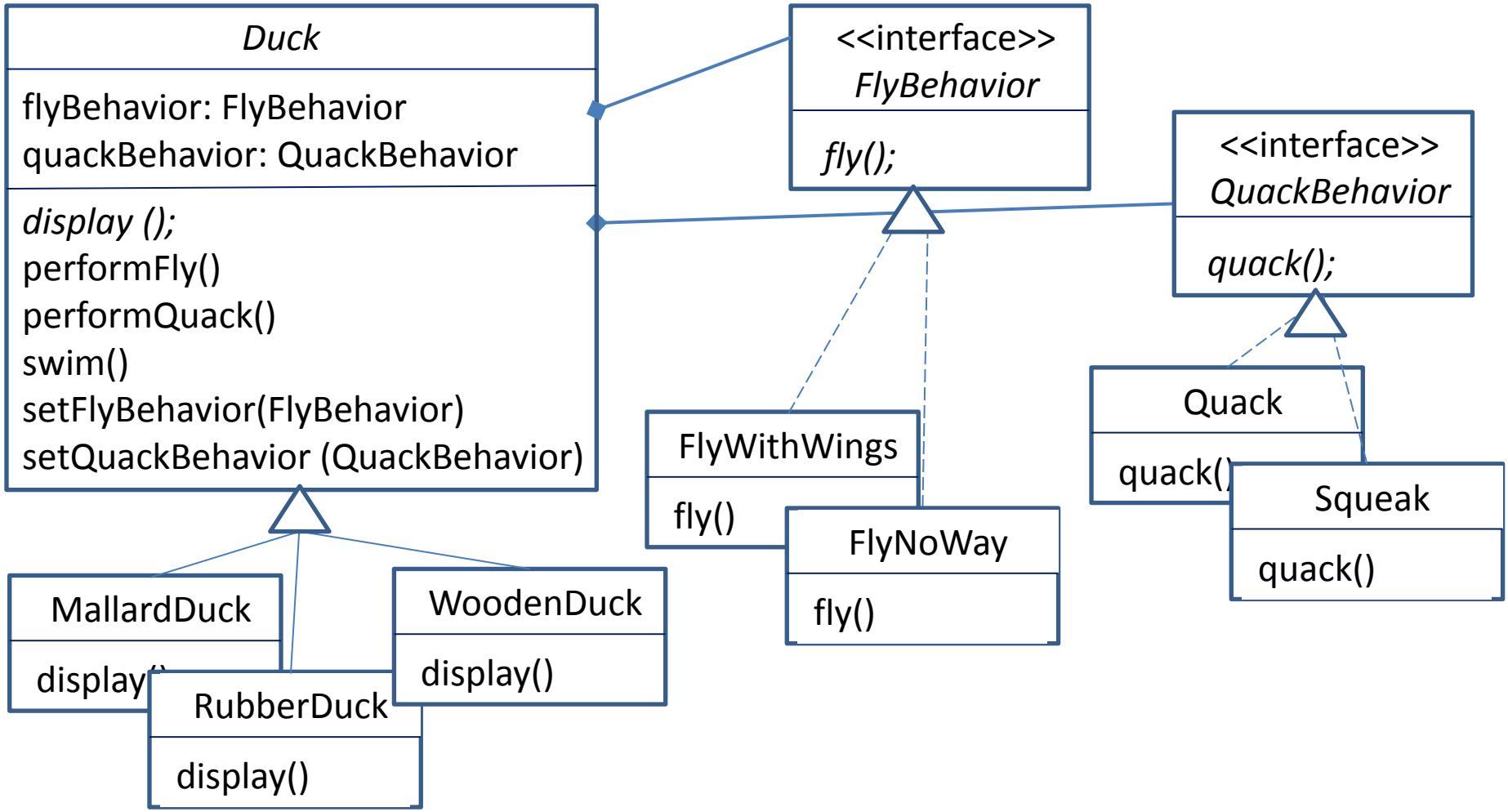
Non-default constructors: reminder

- Game
- BoardGame
- Chess

Final keyword

- Final class attribute – can't change a value (what does it mean for attributes-objects?)
- Final method – can't override a method
  - Private methods are final by default
- Final class – can't extend class

Ducks simulator design



Duck: abstract class with abstract method: display

```
public abstract class Duck {
        FlyBehavior flyBehavior;
        QuackBehavior quackBehavior;

        public Duck() {
        }

        abstract void display();

        public void performFly() {
                flyBehavior.fly();
        }

        public void performQuack() {
                quackBehavior.quack();
        }

        public void swim() {
                System.out.println("All ducks float, even decoys!");
        }
}
```

```
public void setFlyBehavior (FlyBehavior fb) {
                flyBehavior = fb;
        }

public void setQuackBehavior(QuackBehavior qb) {
                quackBehavior = qb;
        }
```

Subclass: MallardDuck

```
public class MallardDuck extends Duck {

        public MallardDuck() {
                quackBehavior = new Quack();
                flyBehavior = new FlyWithWings();
        }

        public void display() {
                System.out.println("I'm a real Mallard duck");
        }
}
```

Subclass: RubberDuck

```
public class RubberDuck extends Duck {

        public RubberDuck() {
                flyBehavior = new FlyNoWay();
                quackBehavior = new Squeak();
        }

        public void display() {
                System.out.println("I'm a rubber duckie");
        }
}
```

Subclass: WoodenDuck

```
public class WoodenDuck extends Duck {
        public WoodenDuck () {
                setFlyBehavior(new FlyNoWay());
                setQuackBehavior(new MuteQuack());
        }
        public void display() {
                System.out.println("I'm a Wooden Decoy duck");
        }
}
```

Interface FlyBehavior

```
public interface FlyBehavior {
        public void fly();
}
```

Implementation of FlyBehavior: FlyWithWings

```
public class FlyWithWings implements FlyBehavior {
        public void fly() {
                System.out.println("I'm flying!!");
        }
}
```

Implementation of FlyBehavior: no fly

```
public class FlyNoWay implements FlyBehavior {
        public void fly() {
                System.out.println("I can't fly");
        }
}
```

Interface: QuackBehavior

```
public interface QuackBehavior {
        public void quack();
}
```

Implementation of QuackBehavior: real quack

```
public class Quack implements QuackBehavior {
        public void quack() {
                System.out.println("Quack");
        }
}
```

Implementation of QuackBehavior: squeak

```
public class Squeak implements QuackBehavior {
        public void quack() {
                System.out.println("Squeak");
        }
}
```

Implementation of QuackBehavior: silence

```
public class MuteQuack implements QuackBehavior {
        public void quack() {
                System.out.println("<< Silence >>");
        }
}
```

Simulator

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

                Duck    mallard = new MallardDuck();
                Duck    rubberDuckie = new RubberDuck();
                Duck    decoy = new WoodenDuck();

                mallard.performQuack();
                rubberDuckie.performQuack();
                decoy.performQuack();

                mallard.performFly();
                rubberDuckie.performFly();
                decoy.performFly();
        }
}
```

Change 1: implementation of new FlyBehavior: Rocket-powered fly

```
public class FlyRocketPowered implements FlyBehavior {
        public void fly() {
                System.out.println("I'm flying with a rocket");
        }
}
```

Change 2: Additional Implementation of new QuackBehavior: Russian quack

```
public class RussianQuack implements QuackBehavior {
        public void quack() {
                System.out.println("Qwa");
        }
}
```

Change 3: new sub-class of Duck

```
public class ModelDuck extends Duck {
        public ModelDuck() {
                flyBehavior = new FlyNoWay();
                quackBehavior = new Quack();
        }

        public void display() {
                System.out.println("I'm a model duck");
        }
}
```

Changing behavior at run tine

```
public class MiniDuckSimulator {

        public static void main(String[] args) {

                ModelDuck      model= new ModelDuck();

                model.performFly();
                model.setFlyBehavior(new FlyRocketPowered());
                model.performFly();
        }
}
```

Actor changes behavior

Strategy design pattern

- Defines a family of algorithms, encapsulates them, and makes them interchangeable by using a common interface
- Strategy lets the algorithm vary independently from clients that use it

OOP Design Principles used in strategy pattern

- Encapsulate what varies and pull it away from what stays the same
- Program to an interface not to an implementation

OOP concepts used in Strategy pattern

- Encapsulation
- Composition
- Inheritance