

Lecture 13 Summary

- Assignment 3
- Polymorphism
- Interfaces

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

1

Assignment 3 Discussion

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

2

By the end of this lecture, you will be able to distinguish between *messages* and *methods* and to use these words to describe code.

You will also be able to create *purely abstract classes* (also called *interfaces*) in Java.

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

3

Polymorphism

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

4

Exercise: card game

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

5

Card.java

```
public class Card
{
    private String suit;
    private short value;

    public Card(String suit, short value)
    {
        this.suit = suit;
        this.value = value;
    }

    public String getSuit()
    {
        return suit;
    }

    public short getValue()
    {
        return value;
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

6

Player.java

```
public abstract class Player
{
    private ArrayList<Card> cards;

    public void addCard(Card card)
    {
        cards.add(card);
    }

    public void removeCard(Card card)
    {
        cards.remove(card);
    }

    public List<Card> getCards()
    {
        return cards;
    }

    public abstract void passHighestCard(Player player);
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

7

Beginner.java

```
public class Beginner extends Player
{
    public void passHighestCard(Player player)
    {
        Card card = getCards().get(0);
        removeCard(card);
        player.addCard(card);
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

8

HonestPlayer.java

```
public class HonestPlayer extends Player
{
    public void passHighestCard(Player player)
    {
        Card maxCard = null;
        for (Card card : getCards())
        {
            if (maxCard == null)
            {
                maxCard = card;
            }
            else if (card.getValue() > maxCard.getValue())
            {
                maxCard = card;
            }
        }

        removeCard(maxCard);
        player.addCard(maxCard);
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

9

Cheater.java

```
public class Cheater extends Player
{
    public void passHighestCard(Player player)
    {
        Card minCard = null;
        for (Card card : getCards())
        {
            if (minCard == null)
            {
                minCard = card;
            }
            else if (card.getValue() < minCard.getValue())
            {
                minCard = card;
            }
        }

        removeCard(maxCard);
        player.addCard(maxCard);
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

10

Language

- so far: “invoke the `passHighestCard` method against an instance of a `Player` object”
- better: “pass the **message** ‘`passHighestCard`’ to an instance of a `Player` object”

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

11

Our exercise

- I sent the **message** “`passHighestCard`” with the parameter “person to your left/right”
 - I can do this without knowing what kind of player you are!
 - The same is true for the *compiler*
- You received the message “`passHighestCard`” and invoked the **method** with the same name in your own definition.
 - this decision is made at *runtime*

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

12

Using the same language, describe...

```
public class SomeProgram
{
    public static void main(String[] args)
    {
        Animal a = new Dog();
        Animal b = new Elephant();
        Animal c = new Human();

        a.speak();
        b.speak();
        c.speak();
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

13

So what is polymorphism?

- poly = many
- morph = form
- polymorphism = many forms
- When a message is sent, the corresponding behaviour can take on many forms.

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

14

Polymorphism Summary

- The compiler knows what kinds of **messages** can be sent to what kinds of objects.
- At runtime, when **messages** are actually sent, the object that receives the message decides what **method** to invoke.
- Polymorphism is the ability for different types of objects to behave differently for the same message.

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

15

Purely Abstract Classes

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

16

Example

```
public class ReallyAbstract
{
    public abstract int method1();
    public abstract int method2();
    public abstract String method3();
    public abstract float method4();
}
```

- What will its subclasses have in common?
- What will they *not* (necessarily) have in common?

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

17

Purely Abstract Class

- No common instance variables
- No common method implementations
- The *only* thing in common are the method **signatures**.
- In Java, this case is considered to be special and is called an **interface**.

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

18

Example

```
public interface ReallyAbstract
{
    public int method1();
    public int method2();
    public String method3();
    public float method4();
}
```

- No instance variables allowed
- No method implementations allowed
- `abstract` keyword unnecessary

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

19

Using Interfaces

```
public class Dog extends Animal implements ReallyAbstract
{
    public int method1()
    {
        // ...
    }

    public int method2()
    {
        // ...
    }

    public String method3()
    {
        // ...
    }

    public float method4()
    {
        // ...
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

20

Why do this?

- A class can only have one superclass (using the `extends` keyword).
- There is **no limit** to how many interfaces a class can have.
- Why wouldn't Java just let you extend from multiple classes?

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

21

Example

```

public interface Transportation {
    public void forward(int speed);
    public void turnLeft(int degrees);
    public void turnRight(int degrees);
    public void stop();
}

public class Car
    implements Transportation {
    // ...
}

public class Elephant extends Animal
    implements Transportation {
    // ... instance variables
    // ... instance methods

    public void forward(int speed) {
        // ...
    }

    public void turnLeft(int degrees){
        // ...
    }

    public void turnRight(int degrees){
        // ...
    }

    public void stop() {
        // ...
    }
}

```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

22

Example: sorting cards

```
public class Card implements Comparable<Card>
{
    private String suit;
    private short value;

    public int compareTo(Card card)
    {
        if (suit.equals(card.suit))
        {
            // use the value to decide order
            if (value < card.value)
                return -1;
            else if (value > card.value)
                return 1;
            else
                return 0;
        }
        else
        {
            // ... order by suit
        }
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

23

Example: sorting cards

```
public abstract class Player
{
    private ArrayList<Card> cards;

    public void sortCards()
    {
        Collections.sort(cards);
    }
}
```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

24

Exercise: what is the output?

```

public class A {
    public void foo() {
        System.out.println("A's foo");
    }
}

public class B extends A {
    public void foo(String bar) {
        System.out.println("B's foo " + bar);
    }
}

public class C extends B {
    public void foo() {
        System.out.println("C's foo");
    }
}

public class Program {
    public static void main() {
        A a1 = new A();
        A a2 = new B();
        A a3 = new C();

        B b1 = new B();
        B b2 = new C();

        C c = new C();

        a1.foo();
        a2.foo();
        a3.foo();

        b1.foo();
        b2.foo();

        c.foo();
        c.foo("bar");
    }
}

```

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

25

Next Class

- Exceptions

March 4, 2009

Slides by Mark Hancock
(adapted from notes by Craig Schock)

26