SWEN20003

Workshop 6, Week 7

Eleanor McMurtry, University of Melbourne

Interfaces

Setting out the framework

- An **interface** specifies methods that an implementing class must have

Setting out the framework

- An **interface** specifies methods that an implementing class must have
- **Weaker** than inheritance: does not imply implementing classes are the “same kind of thing”

An inflexible design

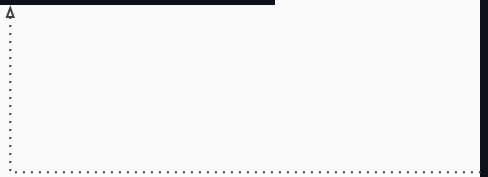
```
public class CounterButton {
    private int count;

    public void click() {
        System.out.println("clicked!");
        ++count;
        System.out.println("count: " + count);
    }
}
```

With an interface

```
public interface Runnable {
    void run();
}
```

With an interface

```java
public interface Runnable {
    void run();
}
```



```java
public class Counter implements Runnable {
    private int count;


    @Override
    public void run() {
        ++count;
        System.out.println("count: " + count);
    }
}
```

Using the `Runnable` interface

- Upcasting!

```
public class Button {
    private final Runnable onClick;

    public Button(Runnable onClick) {
        this.onClick = onClick;
    }

    public void click() {
        System.out.println("clicked!");
        onClick.run();
    }
}
```

Using the `Runnable` interface

- Upcasting!
- Now any action can be performed when the button is clicked.

```
public class Button {
    private final Runnable onClick;

    public Button(Runnable onClick) {
        this.onClick = onClick;
    }

    public void click() {
        System.out.println("clicked!");
        onClick.run();
    }
}
```

Quiz!

- Is `Button` immutable?

```
public class Button {
    private final Runnable onClick;

    public Button(Runnable onClick) {
        this.onClick = onClick;
    }

    public void click() {
        System.out.println("clicked!");
        onClick.run();
    }
}
```

Quiz!

- Is `Button` immutable?
- Only if `onClick` is!

```
public class Button {
    private final Runnable onClick;

    public Button(Runnable onClick) {
        this.onClick = onClick;
    }

    public void click() {
        System.out.println("clicked!");
        onClick.run();
    }
}
```

Aside (not examinable)

- This is an example of a technique called *dependency injection*.

```
public class Button {
    private final Runnable onClick;

    public Button(Runnable onClick) {
        this.onClick = onClick;
    }

    public void click() {
        System.out.println("clicked!");
        onClick.run();
    }
}
```

Interfaces can be used for unrelated classes

- `AudioFile implements Transferable`
- `HttpMessage implements Transferable`
- `DatabaseQuery implements Transferable`

The Comparable interface

```
public class Student implements Comparable<Student> {
    private final int id;
    private final String name;

    public double getWam() { return 80.00; }

    public int compareTo(Student rhs) {
        return id - rhs.id;
    }
```

The Comparable interface

"comparing to Student"

```
public class Student implements Comparable<Student> {
    private final int id;
    private final String name;

    public double getWam() { return 80.00; }

    public int compareTo(Student rhs) {
        return id - rhs.id;
    }
```

Another possible implementation

- Remember to be careful with `doubles`...

```
public int compareTo(Student rhs) {
    double EPS = 1e-2;
    double difference = getWam() - rhs.getWam();

    if (Math.abs(difference) < EPS) {
        return 0;
    } else if (difference < 0) {
        return -1;
    } else {
        return 1;
    }
}
```

Sorting an array of Comparables

```
import java.util.Arrays;

class Program {
    public static void main(String[] args) {
        Student[] students = new Student[] {
                new Student(701212, "Alice"),
                new Student(701010, "Bob"),
                new Student(853535, "Charlie"),
                new Student(142423, "Eve")
        };

        Arrays.sort(students);
        System.out.println(Arrays.toString(students));
    }
}
```