Interface Polymorphism

Ch 4.1-4.5
Topics

1) How can we reduce coupling between classes?
2) How can one piece of code work on different types of objects?
An Interface specifies a set of *public* methods, but..
- It's a contract for providing methods.

```java
public interface LetterGrader {
    String getGrade(double percent);
    double getMinPercentForGrade(String grade);
}
```

"Interface" can refer to two things:
- An interface in Java
  (such as "The LetterGrader interface")
- The..
  (such as "The class's public interface")
To implement an interface, a class must both:

- Say it "implements" the interface
- 

```java
public class EasyLetterGrader implements LetterGrader {
    private static final double BREAK_POINT = 70;

    @Override
    public String getGrade(double percent) {
        if (percent >= BREAK_POINT) {
            return "A+";
        } else {
            return "B";
        }
    }

    @Override
    public double getMinPercentForGrade(String grade) {
        if (grade.compareToIgnoreCase("A+") == 0) {
            return BREAK_POINT;
        } else {
            return 0;
        }
    }
}
```

@Override is an...

Tells Java that this method..
Concrete Types

- **Concrete Type**
  - (not a more general interface or base class).

- **Example**
  - `LetterGrader` is an Interface (not instantiatable), so *not* a concrete type.
  - BAD: `LetterGrader oops = new LetterGrader();`

- **Example**
  - `EasyLetterGrader` is an instantiatable class, so..
  - GOOD: `LetterGrader good = new EasyLetterGrader();`
Polymorphism

- **Polymorphism Example:**
  - A variable of type `LetterGrade` can reference any object of class type which..

  ```java
  LetterGrader g = new EasyLetterGrader();
  computeClassGrades(g);
  LetterGrader g = new HardLetterGrader();
  computeClassGrades(g);
  ```

- **Polymorphism definition:**
  - The exact method to execute is selected at runtime.
  - **Ex:** Does `g.getGrade()` call `EasyLetterGrader.getGrade()`, or `HardLetterGrader.getGrade()`?
Polymorphism Example

class MarkingSystem {
    double[] marks = {74, 85, 25, 55, 93, 1};

    void printLetterGrades() {
        LetterGrader grader = new EasyLetterGrader();
        String[] grades = gradeEachStudent(grader);

        for (String grade : grades) {
            System.out.println("Grade: "+ grade);
        }
    }

    String[] gradeEachStudent(LetterGrader grader) {
        String[] letterGrades = new String[marks.length];
        for (int i = 0; i < marks.length; i++) {
            letterGrades[i] = grader.getGrade(marks[i]);
        }
        return letterGrades;
    }
}

No idea what type of LetterGrader is passed; just that the object..

It can only use..
Terminology

```
«interface»
LetterGrader

MarkingSystem

EasyLetterGrader
CurvedLetterGrader
HardLetterGrader
```
Why Use Polymorphism?

- Exact method (concrete type) determined at runtime.
- works with any object implementing the Interface so independent of object's concrete type.

Design Heuristic:

- Extensible: Reuse code without re-write to support new classes.
Interface Details

• Interface methods are ..
  – can provide “default” implementation of function.

• Can declare.. (automatically public static final)
  
  public interface CardDeck {
    int NUM_CARDS = 52;
    // ...
  }

Interface Details

- An Interface can..

```java
public interface Vehicle {
    void turnTo(double direction);
    void setSpeed(double speedInKmPerH);
}

public interface FlyingVehicle extends Vehicle {
    void flyToAltitude(double altitudeInM);
}
```

- A class implementing `FlyingVehicle` must also implement all of `Vehicle's` methods too.
Exercise

- Which of the following statements work?

```java
public static void main(String[] args) {
    Vehicle v1;
    v1 = new Vehicle();
    v1 = new Car();
    v1 = new Hoverboard();

    FlyingVehicle v2;
    v2 = new Vehicle();
    v2 = new Car();
    v2 = new Hoverboard();

    Car v3;
    v3 = new Vehicle();
    v3 = new Car();
    v3 = new Hoverboard();
}
```
Comparable Review

- Can write algorithms for interface types.

```
interface Comparable<Type> {
    int compareTo(Type obj);
}
```

```java
public class InOrder {
    public static void main(String[] args) {
        Long[] data = new Long[5];
        for (int i = 0; i < data.length; i++) {
            data[i] = i;
        }
        System.out.println("In order? "+ isAscending(data));
    }

    public static boolean isAscending(Comparable[] array) {
        for (int i = 0; i < array.length - 1; i++) {
            Comparable first = array[i];
            Comparable second = array[i+1];
            if (first.compareTo(second) > 0) {
                return false;
            }
        }
        return true;
    }
}
```

This is not quite perfect. **Comparable** is a generic type, so **isAscending()** should have the heading

```java
public static <T extends Comparable<T>> boolean isAscending(T[] array) {
    ...}
```
Comparator Review

- An **idiom** is..
- For creating anonymous classes make a function which creates it.

```java
public interface FileFilter {
    boolean accept(File path);
}

private FileFilter createExtensionFilter() {
    return new FileFilter() {
        @Override
        public boolean accept(File path) {
            return path.isDirectory() || hasAcceptedExtension(path);
        }
    };
}

private void addFolder(File directory) {
    FileFilter filter = createExtensionFilter();
    File[] files = directory.listFiles(filter);
    //..
}
```

Example: As2 solution.
Review Questions

- Can the full type of an object be just an Interface type?
  - No: An object's concrete type cannot be an Interface. An Interface cannot be instantiated, only implemented by other classes.

- Are the following two ideas identical?
  A class which has the same methods as an Interface
  A class which implements the interface?
Summary

- **Interface**: A set of methods & constants.
  - How to define, implement, and use an interface.
- **Concrete Type**: the instantiated type of an object.
- Example uses for **polymorphism**.