Занятие 4. Абстрактные классы и интерфейсы.

- Абстрактные методы, ключевое слово abstract.
- Интерфейсы
- Отделение интерфейса от реализации

Абстрактные классы

- Класс, содержащий абстрактные методы, называется абстрактным классом. Такие классы помечаются ключевым словом *abstract*.
- Абстрактный метод не завершён. Он состоит только из объявления и не имеет тела.
- Нельзя создать экземпляр абстрактного класса.

```
abstract class Figure {
      private String name;

      public Figure(final String name) {
      this.name = name;
      }

     abstract double area();
}
```

Абстрактные классы

```
class Circle extends Figure {

private double radius;

public Circle(final String name, final double radius) {
super(name);
this.radius = radius;
}

@Override
double area() {
return Math.PI * Math.pow(radius, 2);
}
}
```

Интерфейсы

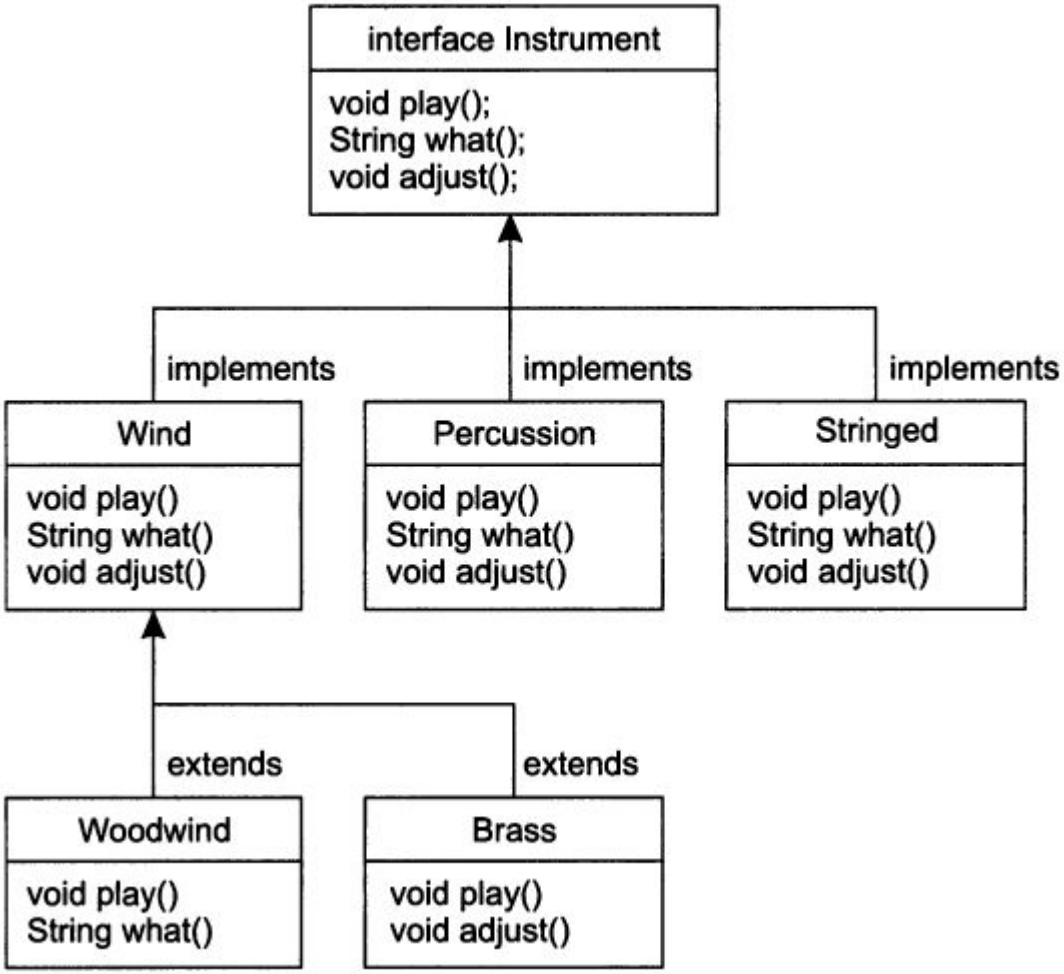
- Интерфейс содержит в себе только сигнатуры методов и статические поля. Помечается ключевым словом `interface`.
- Для использования интерфейса нужно написать его *реализацию* – класс, в котором будут описаны тела методов интерфейса.
- Для реализации интерфейса классом используется ключевое слово `implements`.

```
interface Instrument {
  int VALUE = 5; // static & final
  void play(Note n); // Automatically public
  void adjust();
}
```

Интерфейсы

```
class Wind implements Instrument {
  public void play(Note n) {
    print(this + ".play() " + n);
  }
  public String toString() { return "Wind"; }
  public void adjust() { print(this + ".adjust()"); }
}
```

Интерфейсы


interface Instrument
void play();
String what();
void adjust();
implements
implements
implements
Wind
void play()
String what()
void adjust()
Percussion
void play()
String what()
void adjust()
Stringed
void play()
String what()
void adjust()
extends
extends
Woodwind
void play()
String what()
Brass
void play()
void adjust()

Интерфейсы. Отделение от реализации

```
public interface Processor {
   Object process(Object input);
}
class Upcase implements Processor {
   @Override
   public String process(Object input) { // Covariant return
   return ((String) input).toUpperCase();
   }
}
class Downcase implements Processor {
   public String process(Object input) {
   return ((String) input).toLowerCase();
   }
}
class Splitter implements Processor {
   public String process(Object input) {
   // The split() argument divides a String into pieces:
   return Arrays.toString(((String) input).split(" "));
   }
}
```

Интерфейсы. Отделение от реализации

```
public class Apply {
   // Не важно, объект какого класса передаётся.
   // Главное - он должен реализовывать интерфейс с методом process()
   public static void process(Processor p, Object s) {
   System.out.println("Using Processor " + p.getClass().getSimpleName());
   System.out.println(p.process(s));
   }

   public static String s = "Disagreement with beliefs is by definition
   incorrect";

   public static void main(String[] args) {
   process(new Upcase(), s);
   process(new Downcase(), s);
   process(new Splitter(), s);
   }
}
```

Практическое задание

- Смодулировать товарный склад с возможностью вывода различных отчетов на консоль.

  Создать абстрактный класс `Product` с полями `ean:Long, price:Double, name:String`. От него отнаследовать классы `Food (cal:Int, creationDate:Date, expirationTime:Int)`, `Appliance(inputPower:Int)`, `Clothes(size:Byte, material:String)`. Переопределить метод `toString()` для читаемого вывода.

  Создать массив `Product[] warehouse = {…}`, инициализировать его большим количеством различных товаров. Вывести отчёты:

  1. Отсортировать все элементы по цене, имени.
  2. Сгруппировать по типу и отсортировать в каждой группе по имени и цене.
  3. Отсортировать все элементы `Food` по каллорийности.
  4. Отсортировать все элементы `Appliance` по мощности.
  5. Отсортировать одежду по размеру.

  Создать классы-сервисы с методами для сортировки массивов объектов и интерфейсы для сравнения объектов по различным параметрам.