Short exercise

http://bergel.eu

Pregunta P1

```
class A {
    A() { System.out.println("A.A()"); }
    A(int i) { this(); System.out.println(“A.A(int)”); }
}

class B extends A {
    B() { this(0); System.out.println("B.B()");}
    B(int i) { super(i); System.out.println("B.B(int)");}

    public static void main(String[] argv) {
        new B();
    }
}
```

Pregunta P2

```
class A {
    void bar() { this.foo(); }
    void foo() { System.out.println("A.foo()"); }
}

class B extends A {
    void foo() { System.out.println("B.foo()"); }

    public static void main(String[] argv) {
        new B().bar();
    }
}
```

Pregunta P3

```
class A {
    void bar() { this.foo(); }
    private void foo() { System.out.println("A.foo()");
}

class B extends A {
    void foo() { System.out.println("B.foo()"); }

    public static void main(String[] argv) {
        new B().bar();
    }
}
```

Pregunta P4

```
interface Element {}
class Box implements Element {}

class Canvas {
    public void add(Element e) {
        System.out.println("Element added");
    }
    public void add(Box e) {
        System.out.println("Box added");
    }
}

class Main {
    public static void main(String[] argv) {
        Canvas c = new Canvas();
        Element o1 = new Box();
        Box o2 = new Box();
        c.add(o1);
        c.add(o2);
        c.add(new Box());
    }
}
```

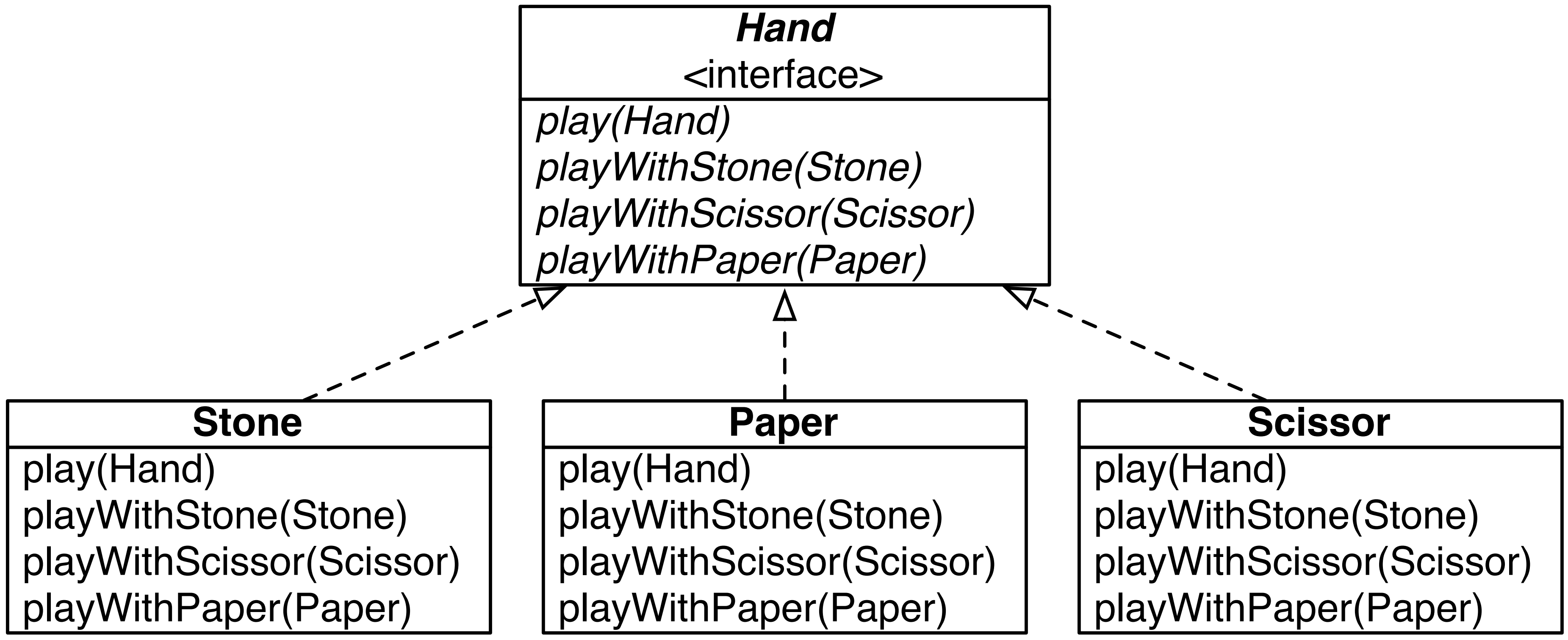
Pierre, feuille, ciseaux

Alexandre Bergel
http://bergel.eu
21/08/2017

Catchipun

Though it looks simple, designing this small game is a fantastic example of the double dispatch design pattern

This pattern is particularly important since it is the base of many other design patterns



Hand
<interface>
play(Hand)
playWithStone(Stone)
playWithScissor(Scissor)
playWithPaper(Paper)
Stone
play(Hand)
playWithStone(Stone)
playWithScissor(Scissor)
playWithPaper(Paper)
Paper
play(Hand)
playWithStone(Stone)
playWithScissor(Scissor)
playWithPaper(Paper)
Scissor
play(Hand)
playWithStone(Stone)
playWithScissor(Scissor)
playWithPaper(Paper)

```
interface Hand {
   // 1 win, 0 draw, -1 loose
   int play (Hand v);
   int playWithStone (Stone stone);
   int playWithPaper (Paper paper);
   int playWithScissor (Scissor scissor);
}
```

For Stone

```
class Stone implements Hand {
  public int play(Hand v) {
    return v.playWithStone (this);
  }
  public int playWithStone (Stone v) {
    return 0;
  }
  public int playWithScissor (Scissor v) {
    return -1;
  }
  public int playWithPaper (Paper v) {
    return 1;
  }
}
```

Paper

```
class Paper implements Hand {
  public int play(Hand v) {
    return v.playWithPaper (this);
  }
  public int playWithStone (Stone v) {
    return -1;
  }
  public int playWithScissor (Scissor v) {
    return 1;
  }
  public int playWithPaper (Paper v) {
    return 0;
  }
}
```

Scissor

```
class Scissor implements Hand {
  public int play(Hand v) {
    return v.playWithScissor (this);
  }
  public int playWithStone (Stone v) {
    return 1;
  }
  public int playWithScissor (Scissor v) {
    return 0;
  }
  public int playWithPaper (Paper v) {
    return -1;
  }
}
```

What you should know!

How does the double dispatch pattern work?

When should one apply this pattern?

What are the benefits when using it?

Can you answer these questions?

Can you give an example where the double dispatch is successfully employed?

Can the double dispatch be used to always get rid of the if statements?

License

http://creativecommons.org/licenses/by-sa/2.5