ohanot

torchat


torchat


torchat


torchat


torchat
At the customer side:

- In the presentation, we introduced the `IPoint` interface and demonstrated 3 implementations for it. 
- We showed that clients dependent on the `IPoint` interface need only to modify the `IPoint` interface implementation, and not recognize the underlying classes that implement them, which makes it easier for changes in the code at a later time.
- Using interfaces saves the double code in the supplier side.

In contrast, when the supplier receives a request for a method from the interface, it gets the class code instead of having to rewrite it. Both suppliers share the same code.

Let's try to identify the redundant code between the 3 implementations of the `IPoint` interface and center the code blocks of each interface in a base class that is shared by all 3 implementations.

**Abstract class: `AbstPoint`**

```java
public abstract class AbstPoint
```

- Class `CartesianPoint`:
  ```java
  public double x() { return x; }
  public double y() { return y; }
  public double rho() { return Math.sqrt(x*x + y*y); }
  public double theta() { return Math.atan2(y,x); }
  ```

- Class `PolarPoint`:
  ```java
  public double x() { return r * Math.cos(theta); }
  public double y() { return r * Math.sin(theta); }
  public double rho() { return r; }
  public double theta() { return theta; }
  ```

- Class `SmartPoint`:

It is difficult to see similarity between the implementations of the methods in this case.

All 4 basic methods have a close relationship to the representation chosen for the fields.
public double distance(IPoint other) {
    return Math.sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
}

public double distance(IPoint other) {
    double deltaX = x() - other.x();
    double deltaY = y() - other.y();
    return Math.sqrt((deltaX * deltaX) + (deltaY * deltaY));
}

public double distance(IPoint other) {
    double deltaX = x() - other.x();
    double deltaY = y() - other.y();
    return Math.sqrt((deltaX * deltaX) + (deltaY * deltaY));
}

public double distance(IPoint other) {
    double deltaX = x() - other.x();
    double deltaY = y() - other.y();
    return Math.sqrt((deltaX * deltaX) + (deltaY * deltaY));
}

public String toString() {
    return "(x=", x, ", y=", y, ", r=", rho(), ", theta=", theta, ")";
}

public String toString() {
    return "(x=", x(), ", y=", y(), ", r=", r, ", theta=", theta, ")";
}
execute the body of the current constructor.