# Factory Method Pattern

Michal Moravik, SD20w2

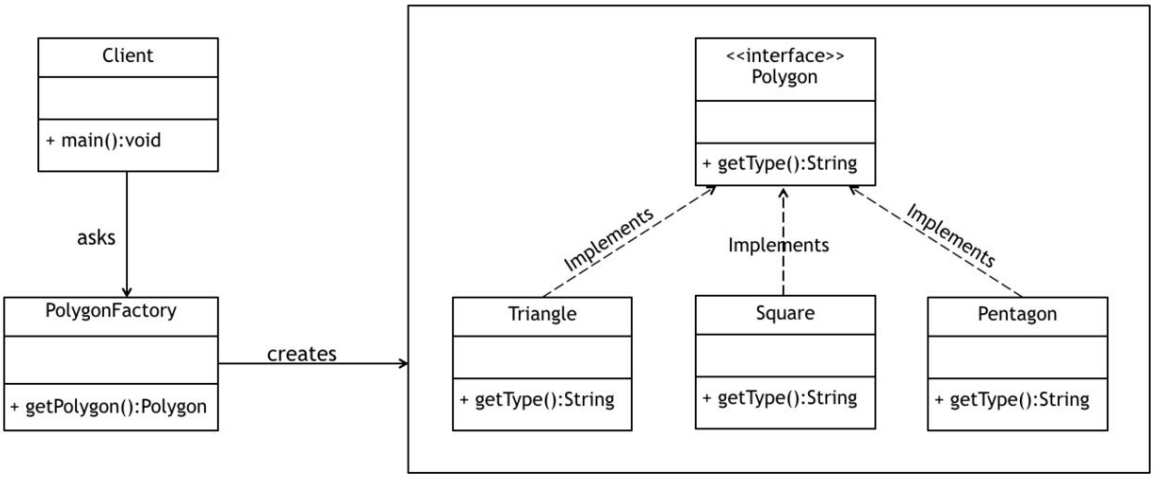
**Name**: Factory Method Pattern

**Category**: Creational pattern

**Intent**: When we want to instantiate one of many different classes that implement (or extend) the same interface (or superclass). We do this in the method called “factory method”. Based on some condition, we can instantiate an object which suits the situation most.

**Motivation:** We want to have classes chosen at runtime so we can switch between them based on the current situation. We want to centralize class selection and we do it in this method.

**UML:**




On this UML we can see that the client asks the polygon factory to create a polygon. Since each class implements a polygon interface, each of them can be spawned after the getPolygon() function is called. Depending on the concrete situation and condition, one of the possible classes will be chosen.

**Implementation:**

```java
public interface Polygon {
    String getType();
}
```

```java
public class Triangle implements Polygon{
    @Override
    public String getType() {
        return "triangle";
    }
}
```

```java
public class Square implements Polygon {
    @Override
    public String getType() {
        return "square";
    }
}
```

```java
public class Pentagon implements Polygon {
    @Override
    public String getType() {
        return "pentagon";
    }
}
```

```java
public class PolygonFactory {
    public static Polygon getPolygon(int numberOfPoints) {
        if (numberOfPoints == 3) {
            return new Triangle();
        } else if (numberOfPoints == 4) {
            return new Square();
        } else {
            return new Pentagon();
        }
    }
}
```

```java
public class ClassNeedingPolygon {
    public String getTypeOfPolygon(Polygon polygon) {
        return polygon.getType();
    }
}
```

```java
    public static void main(String args[]) {
        // the "number of points" could come from multiple sources
        int numberOfPoints = 4;
        // getting the polygon we want to see
        Polygon polygon = PolygonFactory.getPolygon(numberOfPoints);
        // calling method no matter what polygon is instantiated
        // each polygon will return different string
        ClassNeedingPolygon myClass = new ClassNeedingPolygon();
        System.out.println(myClass.getTypeOfPolygon(polygon));
    }
}
```

Because of the number of points specified in “main”, the system will instantiate and use class Square and print out “square” from getType() method.

```
Main (1)
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java
square

Process finished with exit code 0
```

**Consequences:**

**Pros:**

- We have a centralized class selection
- We can decide on an object to instantiate during the runtime
- We can hide the instantiation and we know exactly where this happens
- We are dependent on interfaces (or abstract classes) NOT on concrete classes which gives us a benefit of switching between classes implementing the interface (abstract class) without breaking the code

**Cons:**

- Having a factory for each object type we want to create can introduce too much of complexity to our system
- Code may be harder to read when a lot of it depends on abstractions only

**Known uses:**

- We can use it when we don't know ahead what object we need to instantiate, we will encapsulate the selection of objects, depend on abstraction instead and during the runtime, we decide what exact class we want to instantiate.

**Related patterns:**

- Abstract Factory Pattern