P2 - Exercise hour

Pooja Rani

2020-05-22

Ex. 8 Recap

Identify design patterns

- Builder pattern
- Null Object pattern
- Visitor pattern
- Abstract Factory pattern

Some examples

- Use builder pattern to create complex objects

```
public class PlaintextParser {
     ..
   Game.GameBuilder gameBuilder = new
                    Game.GameBuilder(width, height);
  ..
}

public static class GameBuilder {
     ...
     public GameBuilder(){
          ....
     }
     public Game build(){
               return new Game(this);
     }
}
```

Builder pattern

- Instantiate the game with the data provided by GameBuilder
- GameBuilder is a helper class to create Game instance
- It can validate each Game attribute separately
- Single Responsibility Principle.

Null Object Pattern

- Handle null cases for the objects
- Null object has no side effects as it does nothing
- Used as stub in testing, when certain features such as database is not available for testing

```
public class NullRenderer implements Renderer {
    @Override
      public void render(Game game) { /* do nothing */ }
}
```

Abstract Factory pattern

- Use generic interface to create the related objects

```
public abstract class SokobanObjectProvider {

}
public class TestObjectProvider extends
        SokobanObjectProvider {

}

public class DefaultSokobanObjectProvider extends
         SokobanObjectProvider {

}
```

Abstract Factory pattern

- Open/Closed Principle.
- You can introduce new variants of products without breaking existing client code.

Visitor pattern

- Use the pattern when a behavior makes sense only in some classes of a class hierarchy, but not in others.
- To visit different entities and tiles

```
- EntityVisitor
- TileVisitor
- GameVisitor
```

Visitor pattern

To visit different types of entities such as Box, Player, ExplosiveEntity.

```
public interface EntityVisitor {
    void visitBoxEntity(BoxEntity boxEntity);
    ....
}
public class BoxEntity {
    @Override
    public void accept(EntityVisitor entityVisitor) {
        entityVisitor.visitBoxEntity(this);
    }
}
```

Visitor pattern

To visit different types of tiles such as floor, box, wall.

```
public interface TileVisitor {
    void visitFloorTile(FloorTile floorTile);
    ....
}

public interface GameVisitor extends TileVisitor {
    void visitGame(Game game);
}

public class FloorTile {
    @Override
    public void accept(GameVisitor gameVisitor) {
        gameVisitor.visitFloorTile(this);
    }
}
```

Strategy pattern

- define a family of algorithms and lets you use depending on the object.
- isolate the implementation details of an algorithm from the code that uses it.

```
public interface CollisionVisitor {
        void collideWith(Entity entity, Point delta);
        ..
}

public class BoxEntity {
@Override
    public void collideWith(Entity entity, Point delta) {
        entity.collideWithBoxEntity(this, delta);
    }
}
```

Singleton pattern

- ensure that a class has only one instance

```
public abstract class SokobanObjectProvider {
    protected SokobanObjectProvider() {}

    public static SokobanObjectProvider instance() {
        if (instance == null) {
            instance = defaultInstance();
        }
        return instance;
    }
}
```

Other patterns

- Iterator pattern

Pharo

- Pharo is a dynamic typed language
- Style matches to the natural language, English
- A live programming environment
- Supports live debugging
- Inspect objects with custom representations

Basic blocks

```
2 raisedTo: 30                          "1073741824- "
15 / 25                                  "(3/5)- Fraction"
'Hello Smalltalk'                   "'Hello Smalltalk' -ByteString"
anArray := #(1 2)
```

How do you write Loops?

Java

```
for(int i = 1; i < 10 ; i++)
            System.out.print(i);
```

Pharo

```
(1 to: 9) do: [:x | Transcript show: x printString]
```

Detect first odd number from the array?

Java

```
int[] array = {21, 23, 53, 66, 87};
        Integer result = null;
        for (int i = 0; i < array.length ; i++) {
            if (array[i] \% 2 == 1) {
                  result = array[i];
                  break;
} }
        if (result == null)
            throw new Exception(?Not found?);
```

Pharo

```
#(21 23 53 66 87) detect: [ :x | x odd]
```

Note: Note that arrays are 1-based-
that is, the first valid index is 1, rather than 0.

Exercise 10

- Revisit Turtle game from exercise 3
- Move turtle using 4 commands
- Commands are already created
- Understand ‘TurtleModel‘ and ‘BoardModel‘ class and document the classes

Document the classes

- Document all the details like purpose of the classes, what they do, instance variables, APIs warnings, observations etc. that you think is important to understand and extend these classes
- Pharo use Class comments as a primary source to document all such details
- Write all the details in comments
- Document ‘TurtleModel‘ and ‘BoardModel‘ class and document the classes

NOTE

- There are 2 exercise patterns, solve according to your group number.
- Differences in the class comment template
- See *exercise_09.md* for more details
- Deadline 29th May, 2020