P2: Coding Issues & Pool Hour

Claudio Corrodi

April 27 2018

Coding Issues: Attributes

```
class Board {
    public Square firstSquare;
}

class Game {
    public void client() {
         Square start = board.firstSquare;
         // ...
    }
}
```

Coding Issues: Attributes

```
class Board {
    public Square firstSquare;
}

class Game {
    public void client() {
          Square start = board.firstSquare;
          // ...
    }
}
```

What if we change “firstSquare”?

Coding Issues: Attributes

```
class Board {
    public List<Square> squares;
}

class Game {
    public void client() {
         Square start = game.squares.get(0);
         // ...
    }
}
```

What if we change “firstSquare”?

Does not work anymore! We need to change code in all clients!

Coding Issues: Attributes

```
class Board {
    protected Square firstSquare;

    public Square getFirstSquare() {
        return firstSquare;
    }
    public void setFirstSquare(Square aSquare) {
        firstSquare = aSquare;
    }
}

private void client() {
    Square start = board.getFirstSquare();
    // …
}
```

With getters/setters, we can change the implementation without affecting clients.

Coding Issues: Attributes

```
class Board {
    protected List<Square> squares;

    public Square getFirstSquare() {
        return squares.get(0);
    }
    public void setFirstSquare(Square aSquare) {
        squares.set(0, aSquare);
    }
}

private void client() {
    Square start = board.getFirstSquare();
    // …
}
```

With getters/setters, we can change the implementation without affecting clients.

Coding Issues: Attributes

- Make attributes protected
  - Subclasses should be able to access own state
- Use getters and setters to make them available to clients
  - Does not expose raw data structures
  - We can increase the complexity of getters and setters without worrying about clients

Coding Issues: Constants

```
public class Board {
    protected final int BOARD_SIZE;
    protected final char[] ROW_NAMES = { 'A', 'B', 'C' };
    protected final int[] COL_NAMES = { 1, 2, 3};
}
```

Coding Issues: Constants

```
public class Board {
    protected final int BOARD_SIZE;
    protected final char[] ROW_NAMES = { 'A', 'B', 'C' };
    protected final int[] COL_NAMES = { 1, 2, 3};
}
```

These are not constants

Coding Issues: Constants

```
public class Board {
    protected final int BOARD_SIZE;
    protected final char[] ROW_NAMES = { 'A', 'B', 'C' };
    protected final int[] COL_NAMES = { 1, 2, 3};
}
```

These are not constants

```
public class Board {
    protected final int boardSize;
    protected final char[] rowNames = { 'A', 'B', 'C' };
    protected final int[] colNames = { 1, 2, 3};
}
```

Use camelCase for attributes

Coding Issues: Constants

```
public class Board {
    protected final int BOARD_SIZE;
    protected final char[] ROW_NAMES = { 'A', 'B', 'C' };
    protected final int[] COL_NAMES = { 1, 2, 3};
}
```

These are not constants

```
public class Board {
    protected final int boardSize;
    protected final char[] rowNames = { 'A', 'B', 'C' };
    protected final int[] colNames = { 1, 2, 3};
}
```

Use camelCase for attributes

```
public class Board {
    protected static final int BOARD_SIZE = 3;
    protected static final char[] ROW_NAMES = { 'A', 'B', 'C' };
    protected static final int[] COL_NAMES = { 1, 2, 3 };
}
```

’static final’ for constants

Coding Issues: Constants vs enumerations

```
final class Direction {
    public static final int LEFT = 1;
    public static final int RIGHT = 2;
    public static final int UP = 3;
    public static final int DOWN = 4;
}
public static Command createCommand(int type) {
    if (type == LEFT) {
        return new CommandLeft();
    } else if (type == RIGHT) {
        return new CommandRight();
    } else {
        // ...
    }
    return null;
}
```

Coding Issues: Constants vs enumerations

```java
final class Direction {
    public static final int LEFT = 1;
    public static final int RIGHT = 2;
    public static final int UP = 3;
    public static final int DOWN = 4;
}
public static Command createCommand(int type) {
    if (type == LEFT) {
        return new CommandLeft();
    } else if (type == RIGHT) {
        return new CommandRight();
    } else {
        // ...
    }
    return null;
}
```

Lots of “if-then-else” statements. Code smell!

Coding Issues: Constants vs enumerations

```
enum Direction {
    LEFT,
    RIGHT,
    UP,
    DOWN
}
Command createCommand(Direction dir) {
    switch (dir) {
        case LEFT: return new CommandLeft();
        case RIGHT: return new CommandRight();
        case UP: // ...
        case DOWN: // ...
    }
    // ...
}
```

Coding Issues: Constants vs enumerations

```
enum Direction {
    LEFT,
    RIGHT,
    UP,
    DOWN
}
Command createCommand(Direction dir) {
    switch (dir) {
        case LEFT: return new CommandLeft();
        case RIGHT: return new CommandRight();
        case UP: // ...
        case DOWN: // ...
    }
    // ...
}
```

Slightly better, less error prone.

Coding Issues: Constants vs enumerations

```
interface CommandFactory {
    Command create();
}
enum Direction implements CommandFactory {
    LEFT {
        public Command create() {
            return new CommandLeft();
        }
    },
    RIGHT {
        public Command create() {
            return new CommandRight();
        }
    },
    // ...
}
```

Enums can implement interfaces.

Coding Issues: Constants vs enumerations

```
interface CommandFactory {
    Command create();
}
enum Direction implements CommandFactory {
```

```
// Client
Command createCommand(Direction dir) {
    return dir.create();
}
```

```
    },
    RIGHT {
        public Command create() {
            return new CommandRight();
        }
    },
    // ...
}
```

Enums can implement interfaces.

Coding Issues: Switch instructions

```
private int convertToInt(char c) {
    int output;
    switch (c) {
        case 'a': output = 0;
        case 'b': output = 1;
        case 'c': output = 2;
        case 'd': output = 3;
        case 'e': output = 4;
        case 'f': output = 5;
        case 'g': output = 6;
        case 'h': output = 7;
        case 'i': output = 8;
        case 'j': output = 9;
        default: output = 10;
    }
    return output;
}
```

What does convertToInt('e') return?

Coding Issues: Switch instructions

```
private int convertToInt(char c) {
    int output;
    switch (c) {
        case 'a': output = 0;
        case 'b': output = 1;
        case 'c': output = 2;
        case 'd': output = 3;
        case 'e': output = 4;
        case 'f': output = 5;
        case 'g': output = 6;
        case 'h': output = 7;
        case 'i': output = 8;
        case 'j': output = 9;
        default: output = 10;
    }
    return output;
}
```

Always prints 10!

What does convertToInt('e') return?

Coding Issues: Switch instructions

```
private int convertToInt(char c) {
    int output;
    switch (c) {
        case 'a': output = 0; break;
        case 'b': output = 1; break;
        case 'c': output = 2; break;
        case 'd': output = 3; break;
        case 'e': output = 4; break;
        case 'f': output = 5; break;
        case 'g': output = 6; break;
        case 'h': output = 7; break;
        case 'i': output = 8; break;
        case 'j': output = 9; break;
        default: output = 10; break;
    }
    return output;
}
```

Don’t forget to break or return

Coding Issues: Switch instructions

```
private boolean isLowercaseLetterBeforeE(char c) {
    boolean result;
    switch (c) {
        case 'a':
        case 'b':
        case 'c':
        case 'd':
            result = true;
            break;
        default:
            result = false;
            break;
    }
    return result;
}
```

“Falling through” can be useful…

Coding Issues: Switch instructions

```
private boolean isLowercaseLetterBeforeE(char c) {
    return c - 'a' < 4;
}
```

This is a bit simpler though

Coding Issues: Switch instructions

```java
private boolean isLowercaseLetterBeforeE(char c) {
    return c - 'a' < 4;
}
```

But is it a good implementation?

Coding Issues: Switch instructions

```
private boolean isLowercaseLetterBeforeE(char c) {
    assert c >= 'a' && c <= 'z';
    return c - 'a' < 4;
}
```

Better?

Coding Issues: Switch instructions

```java
/**
 * Checks whether the given character comes
 * before 'e' in the alphabet.
 * @param c a character, must be a lowercase
 * letter between 'a' and 'z'
 */
private boolean isLowercaseLetterBeforeE(char c) {
    assert c >= 'a' && c <= 'z';
    return c - 'a' < 4;
}
```

Don’t forget your contracts

Remaining exercises

- 3 more exercises
- Exercise 8: two weeks, mandatory
- Exercises 7 and 9: one week
- **If you pass exercises 1–6 and 8,**
  you can skip exercise 7 or 9
- Otherwise, you must hand in all exercises