AARHUS UNIVERSITET

Software Engineering and Architecture

Comments on Mandatory and Stuff

AARHUS UNIVERSITET

Name Booleans?

```
boolean listHasCities = cities.size() > 0;
if (!listHasCities) return null;
```

eller ville

```
if(!cities.size() > 0) return null;
```

Learning = overdo it ☺

Or why not:
// Test that there are cities in list
If (!cities.size() > 0) return null;

No, comments get out of date quickly!

AARHUS UNIVERSITET

WinnerStrategies and State

ZetaCiv is a pain…

AARHUS UNIVERSITET

Problem statement

- When doing compositional designs we
  - Cut behavior that is often otherwise in the *same* abstraction, and therefore have **access to the same state (i.e. The same set of instance variables)...**
    - Parametric: all in the same class
    - Polymorphic: subclasses still contain all of superclass var.
  - ... into multiple objects, like A and B
- **But how do they share/exchange state?!?**
  - Some in A, some in B? All in B? All in A? Or in C???

Example

- WinnerStrategy: Algorithm to determine winner
- Based upon knowledge of
  - Alpha: The **age** (correlates to rounds)
  - Beta: Owner of all **cities**
  - Epsilon: **Three attacks** won by **player**
  - **Zeta**: Rounds+Beta+Epsilon'
    - Epsilon' = counting won attacks only starts after 20 rounds

> *Winner.* The winner is the player that first conquers all cities in the world (like BetaCiv). However, in case the game lasts more than 20 rounds then the winner is the first player to win three attacks (like EpsilonCiv). The counting of attacks won does not start until the 20th round has ended.

- But who is responsible for "knowing it"?

Example

- Who should know it? I.e. have the state variable?
  - Game or the WinnerStrategy or some third object?
  - Number of **rounds** played/age
  - Owner of all **cities**
  - **Attacks** won by **player**
  - Counting only starts after 20 rounds
- Obviously a responsibility of Game to know
- Not obviously something that Game should care about…

AARHUS UNIVERSITET

The options

- What are our options
  - A) Game / GameImpl
    - Putting state information there which is variant specific
      - That is: Only demanded by a single or two variants
    - Lowering cohesion
    - *And the big issue is how to handle 'reset' in ZetaCiv after 20 rounds?*
  - B) Strategies
    - Putting *variant specific* state information in the variant strategy
      - High cohesion
    - But – We then have to tell the strategies
  - C) Move responsibility to a third object
    - Game *has-a* GameStatistics object

AARHUS UNIVERSITET

A) In Game

- Put all state in Game, even if only used by one variant
  - *Low cohesion*
  - *The same effect as the 'bubbling up the inheritance hierarchy' in the inhertiance-based discussion of State pattern*
  - GameImp becomes a **Blob class** with all sorts of unrelated stuff

- Conclusion:
  - Ok to have state that makes sense for all/most variants in Game
  - Not so for very specific state information

AARHUS UNIVERSITET

B) In the Strategies

- *Say we store it in the strategies (battles won)*
- How to *tell the strategy when battle is won*, then?
  - A) Add methods to WinnerStrategy
    - incrementBattlesWonBy(Player p)
    - incrementRoundsPlayed();
  - B) Make WinnerStrategy *observer* on game events
    - WinnerStrategy extends GameObserver
      - battleWonEvent(Player p);
      - roundEndEvent();
- Both solutions require that Game *informs* strategy object
  - If (battleWon) {winnerstrategy.incrementBattlesWonBy(player);}

AARHUS UNIVERSITET

B) Resetting

- ZetaCiv just delegates to its ‘currentState’
  - isBattleWon(…) { currentState.isBattleWon(…); }

- Means resetting battle counter **is easy**
  - In betaState it forwards request to beta’s battle counting method
    - Which does nothing!
  - Only in epsilonState does it forward to epsilon’s state
    - And as it changes state only 20 rounds, no calls are made before that; and epsilon only starts counting then!

AARHUS UNIVERSITET

C) Exploring the GameStatistics

- I implemented the GameStatistics option one year…
  - int w = game.getStatistics().getBattlesWonBy(Player.GREEN);
  - Game will increment this counter every time a battle is won

- Reset becomes a headache!
  - Do not want to reset the counters after 20 rounds; it is then not game's statistics but just a silly strategy's statistics
  - Solve it by providing epsilon's winner strategy with a *decorated game instance*
    - *We will talk about Decorator Pattern later…*

AARHUS UNIVERSITET

Tedious…

- Conclusion:
  - This is a *Do Over*

```
public ZetaCivWinnerStrategy() {
  super();
  betaState = new ConquerAllCitiesWinnerStrategy();
  epsilonState = new FirstToWinThreeBattlesWinnerStrategy();
  currentState = betaState;
}
```

```
@Override
public Player calculateWinner(Game game) {
  // State change testing; actually there are three
  // states: beta, changeToEpsilon, and epsilon state
  GameStatistics stats = game.getStatistics();
  if (stats.getRoundCounter() <= 20) {
    currentState = betaState;
  } else if (stats.getRoundCounter() == 21) {
    currentState = epsilonState;
    // Battles won only starts counting now!
    // We handle this by making a copy of the
    // game statistics at this moment in time
    // and then construct a new one by
    // subtracting the current battles won
    // score
    gameStatsAt21round = (GameStatistics) game.getStatistics().clone();
  } else {
    currentState = epsilonState;
    // We have to make the Game return a
    // modified statistics object; and do this
    // using a Decorator
    GameStatistics modifiedStats = computeResetStatistics(game.getStatistics());
    game = new DecoratedStatisticsGame(game, modifiedStats);
  }
  // Finally, just delegate to the current state
  return currentState.calculateWinner(game);
}
```

AARHUS UNIVERSITET

So – a Conclusion?

- I would definitely go for option B)
  - *Put highly variant specific state in the variant specific strategies/delegates*
  - *Add code in GameImpl to tell delegates about state changes*
    - *Direct method calls or Observer pattern*
  - *Regarding Observer pattern*
    - *A bit of an over-engineered solution (You Ain't Gonna Need It)*
    - *UNLESS you can see obvious other benefits of the approach*
      - *Like have a in-game statistics system that lists all major battles, events, in the game, as indeed Civ games often have…*

AARHUS UNIVERSITET

Similar Question w. Methods

- Who has the ‘getFriendlySupport()’ method?
  - Used to calculate winners of attacks in EpsilonCiv
    - (and thus also in ZetaCiv)
- We may choose
  - A) Game
  - B) Strategy
  - C) A third object (Utility)
- Similar arguments
  - A) leads to Blob class ☹
  - B) high cohesion (especially if only used in *one* variant/strategy)
  - C) I choose this if 1) used in *multiple* places and 2) easy to test

AARHUS UNIVERSITET

Context Information

A general approach

(Remember: consider cost and benefits!)

AARHUS UNIVERSITET

Context for Strategies

- Source: *Finn Rosenbech Jensen, former TA, 2009*

- The problem we have been struggling with
  - WinnerStrategy: Have to use all kinds of info from Game
    - **Shall we pass ”Game” or ”GameImpl” ?**

```
public interface WinnerStrategy {
  public Player getWinner(Game game);
}
```

  - UnitActionStrategy: Have to *modify* game
    - Shall we pass data structures back and forth?
    - Shall we introduce new mutator methods in GameImpl?

AARHUS UNIVERSITET

For Modification

- For strategies that *modify game state* we have options
- A) Pass GameImpl, let the datastructures be package visible, put strategies in same package
  - Create city: gameimpl.cityMap.put(pos, new City(…))
  - *Bad: Tight coupling to specific datastructures*
- B) Pass GameImpl, have mutator methods *not* in Game interface
  - Create city: gameimpl.createCityAt(pos, new City(…))
  - *Much better: No datastructure coupling but still coupling to GameImpl, meaning it cannot be changed*

AARHUS UNIVERSITET

For Modification

- For strategies that *modify game state* we have options
- C) Just pass ‘dumb’ data structures back to Game to be interpreted
  - Action action = actionStrategy.performActionAt(pos, unit);
  - If (action.city != null) { createCityAt(action.cityPos, action.city); }
  - Hm: IMO just a cumbersome way of doing option B)

AARHUS UNIVERSITET

For Modification

- For strategies that *modify game state* we have options
- D) Introduce 'ModifiableGame' with mutators
  - Interface ModifiableGame extends Game {
  - boolean createCityAt(Position p, Player owner);
  - }
  - *GameImpl implements ModifiableGame*
  - Then ActionStrategy's declaration becomes
    - performUnitAction(ModifiableGame mgame, Position p, Unit u)
  - And usage is just in a given implementation is then
    - Mgame.createCityAt(…);
  - Good: UnitActionStrategy decoupled from datastructures AND GameImpl
  - Bad: Yet another interface to consider…

AARHUS UNIVERSITET

Jensen’s Note

AARHUS UNIVERSITET

Private Interface

- James Newkirk pattern
  - **Private interface**
    - Responsibility: Publish just the context information for WinnerStrat.

```
/** Publishes the methods available for concrete WinnerStrategy classes.
*   This is an application of "Private Interface" Pattern.
*/
public interface WinnerStrategyContext {
  public int getAge();
  public Collection<Player> getOwners();
}
```

  - Only provide that to the strategy

```
public interface WinnerStrategy {
  public Player getWinner(WinnerStrategyContext context);
}
```

How to Call?

```
public class GameImpl implements Game {
  ...
  public Player getWinner() {
    return _winnerStrategy.getWinner(new WinnerContext() {
          public int getAge() {
            return GameImpl.this.getAge();
          }

          public Collection<Player> getOwners() {
            ArrayList<Player> result = new ArrayList<Player>();
            result .add(_redCity.getOwner());
            result .add(_blueCity.getOwner());
            return result;
          }
        });
  }
```

Or:

b) Have a private inner class

c) Let 'GameImpl implements Game, WinnerContext'

AARHUS UNIVERSITET

Benefits/Liabilities

- Benefits
  - Better decoupling
    - Strategies are not coupled to game, i.e. cannot invoke ‘moveUnit()’
  - No bloating of Game/GameImpl with new methods

- Liabilities
  - Yet another interface to overview
  - Constant modifications to WinnerStrategyContext as we add more and more winner strategies

AARHUS UNIVERSITET

Interface Segregation

- Observation (Robert Martin)

> Many client specific interfaces are better than one general purpose interface.

> Clients should not be forced to depend on interfaces they don't use.

- Cut Game into two: Query and Command interface

```
public interface GameContext {
  public Tile getTile( Position p );
  public Unit getUnitAt( Position p );
  public City getCityAt( Position p );
  public Player getPlayerInTurn();
  public Player getWinner();
  public int getAge();
}
```

```
public interface Game extends GameContext {
    Command methods / mutators here
}
```

AARHUS UNIVERSITET

Can then be extended

- Public interface GameInternalManipulor extends Game
  - public void createCityAt(Position p);
  - public void createUnitAt(Position p);
  - }

- Which GameImpl then implements and then
  - UnitActionStategy { performAction(GameInternalManipulator g);}

- Now GammaCivUnitActionStrategy can create city as it has access to the ‘createCityAt()’ method

Benefits/Liabilities

- Benefits
  - High cohesion in each of the interfaces
  - Low coupling
    - Each interface only talks with *just enough* game behaviour
- Liabilities
  - But it comes at the cost of *many more interfaces*
  - *An each new feature probably still requires 'change by modification'*
    - That is, the same cost as if I had passed the GameImpl
- Conclusion: It depends ☺

Last Year

AARHUS UNIVERSITET

‘Do not talk to strangers’

- *If you talk through a chain of objects, you have high coupling which is bad*
  - Do not have 2+ ‘dots’ in any statement
    - x.getY().getZ().getS().getT().getU().doThing()
- What about Java Stream API then???

```
  List<Integer> numbers = new ArrayList<>();
numbers.addAll(Arrays.asList(1, 20, 3, 10, 20, 30, 4, 50, 80, 1, 2));

List<String> number_str = numbers.stream()
        .filter(num -> num >= 10)//check num greater than 10
        .limit(5)//stop loop at 5
        .sorted()//sort the list
        .map(num -> String.format("Number %d", num))//typecast into String List
        .collect(Collectors.toList());
```

Exceptions and Error Codes

- Question from Student
  - *Modern code have stuff like Optional<T> and Result<Val,Code> to be returned; are exceptions not old fashioned?*
  - HTTP calls return also a multivalued object
    - Body and Status Code, like **404 not found** or **200 OK**
- Still: ***Prefer exceptions because***
  - Exceptions travel up the call stack ***without testing at all levels!***

```
java.lang.NullPointerException
    at org.apache.tools.ant.taskdefs.ExecuteJava.execute(ExecuteJava.java:19
    at org.apache.tools.ant.taskdefs.Java.run(Java.java:772)
    at org.apache.tools.ant.taskdefs.Java.executeJava(Java.java:222)
    at org.apache.tools.ant.taskdefs.Java.executeJava(Java.java:136)
    at org.apache.tools.ant.taskdefs.Java.execute(Java.java:109)
    at org.apache.tools.ant.UnknownElement.execute(UnknownElement.java:293)
    at sun.reflect.GeneratedMethodAccessor4.invoke(Unknown Source)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.apache.tools.ant.dispatch.DispatchUtils.execute(DispatchUtils.jav
    at org.apache.tools.ant.Task.perform(Task.java:348)
    at org.apache.tools.ant.Target.execute(Target.java:435)
    at org.apache.tools.ant.Target.performTasks(Target.java:456)
    at org.apache.tools.ant.Project.executeSortedTargets(Project.java:1405)
    at org.apache.tools.ant.Project.executeTarget(Project.java:1376)
    at org.apache.tools.ant.helper.DefaultExecutor.executeTargets(DefaultExe
    at org.apache.tools.ant.Project.executeTargets(Project.java:1260)
    at org.apache.tools.ant.Main.runBuild(Main.java:853)
    at org.apache.tools.ant.Main.startAnt(Main.java:235)
    at org.apache.tools.ant.launch.Launcher.run(Launcher.java:285)
    at org.apache.tools.ant.launch.Launcher.main(Launcher.java:112)
Caused by: java.lang.NullPointerException
    at java.io.StringReader.(StringReader.java:50)
    at org.json.simple.parser.JSONParser.parse(JSONParser.java:79)
    at org.json.simple.parser.JSONParser.parse(JSONParser.java:75)
    at cloud.cave.config.socket.SocketClientRequestHandler.sendRequestAndBlo
    at cloud.cave.client.ClientCommon.requestAndAwaitReply(ClientCommon.java
    at cloud.cave.client.PlayerProxy.requestAndAwaitReply(PlayerProxy.java:2
    at cloud.cave.client.PlayerProxy.getWeather(PlayerProxy.java:210)
    at cloud.cave.client.CmdInterpreter.handleMultipleCharCommand(CmdInterpr
    at cloud.cave.client.CmdInterpreter.readEvalLoop(CmdInterpreter.java:98)
    at cloud.cave.main.CaveCmd.main(CaveCmd.java:36)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
```