AARHUS UNIVERSITET

Software Engineering and Architecture

Observations On Mandatory From TAs

AARHUS UNIVERSITET

Use {} always

- Why is the **second** form the one to prefer?

```
if (true) doSomething();
else {
  doSomethingElse();
}
```

over

```
if (true) {
 doSomething();
} else {
  doSomethingElse();
}
```

AARHUS UNIVERSITET

Define Unit Constants

- One issue that pops up in HotCiv is all those parameters that units need to have defined...
  - Archer: Defense 3, attack 2, move count 1, cost 10
  - Bomb: Defense 1, attack 0, move count 2, cost 60

- How to do that in a flexible way?

AARHUS UNIVERSITET

One Attempt

```
public interface UnitConstantsStrategy {
    int getDefaultMoveCount(String type);
    int getDefensiveStrength(String type);
    int getAttackingStrength(String type);
    int getCost(String type);
    boolean canMoveTo(String unitType, String tileType);
}
```

Idea:
One abstraction has all params.
Delegate to parameterized methods in GameConstants

```
public class AlphaCivUnitConstantsStrategy implements UnitConstantsStrategy {
    @Override
    public int getDefaultMoveCount(String type) {
        return 1;
    }

    @Override
    public int getDefensiveStrength(String type) {
        return GameConstants.getDefensiveStrength(type);
    }

    @Override
    public int getAttackingStrength(String type) {
        return GameConstants.getAttackingStrength(type);
    }

    @Override
    public int getCost(String type) {
        return GameConstants.getCost(type);
    }

    @Override
    public boolean canMoveTo(String unitType, String tileType) {
        boolean movingToOcean = tileType.equals(GameConstants.OCEANS);
        boolean movingToMountain = tileType.equals(GameConstants.MOUNTAINS);
        return !(movingToMountain || movingToOcean);
    }
}
```

And the Bomb

```
public class ThetaCivUnitConstantsStrategy implements UnitConstantsStrategy {
    private UnitConstantsStrategy baseStrategy = new AlphaCivUnitConstantsStrategy();

    @Override
    public int getDefaultMoveCount(String type) {
        if (type.equals(ThetaCivFactory.BOMB)) return 2;
        return baseStrategy.getDefaultMoveCount(type);
    }

    @Override
    public int getDefensiveStrength(String type) {
        if (type.equals(ThetaCivFactory.BOMB)) return 1;
        return baseStrategy.getDefensiveStrength(type);
    }

    @Override
    public int getAttackingStrength(String type) {
        return baseStrategy.getAttackingStrength(type);
    }

    @Override
    public int getCost(String type) {
        if (type.equals(ThetaCivFactory.BOMB)) return 60;
        return baseStrategy.getCost(type);
    }

    @Override
    public boolean canMoveTo(String unitType, String tileType) {
        if (unitType.equals(ThetaCivFactory.BOMB)) return true;
        return baseStrategy.canMoveTo(unitType, tileType);
    }
}
```

Idea:
ThetaCiv *is-a* AlphaCiv

Issues?

AARHUS UNIVERSITET

Strategy = algorithm

- Strategy handles *algorithms*. There is no algorithm here.

- It is just a question of setting parameters in units. So use a datastructure to do that.

- *For dumb data, use a data oriented programming way.*

AARHUS UNIVERSITET

Data is Data!

- A old school C way

```
public static void main(String[] args) {
  System.out.println("== Demo Unit parameterization ==");

  // Format: [defense, attack, move, cost]
  int[] archerStats = new int[] {3, 2, 1, 10};
  int[] bombStats   = new int[] {1, 0, 2, 60};

  Unit archer = new StandardUnit(GameConstants.ARCHER,
                                 Player.RED,
                                 archerStats);
  dumpStats(archer);
```

AARHUS UNIVERSITET

Indexing

- Can be done in many ways. Here, I am using Enum

```
class StandardUnit implements  Unit {
  private String type;
  private Player owner;
  private int[] stats;

  enum index { DEFENSE, ATTACK, MOVE, COST };

  public StandardUnit(String type, Player owner, int[] stats) {
    this.type = type;
    this.owner = owner;
    this.stats = stats;
  }
  public String getTypeString() { return type; }
  public Player getOwner() { return owner; }
  public int getMoveCount() { return stats[index.MOVE.ordinal()]; }
  public int getDefensiveStrength() { return stats[index.DEFENSE.ordinal()]; }
  public int getAttackingStrength() { return stats[index.ATTACK.ordinal()]; }
}
```

- *Perhaps, use 'final int DEFENSE = 0;' instead...*

AARHUS UNIVERSITET

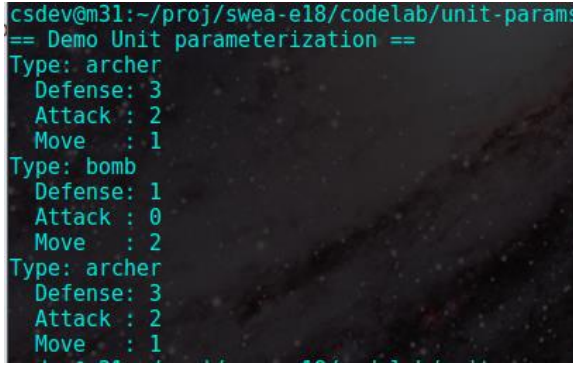
Upping the Dynamics

- But – if the unit type is given dynamically...

```
// More Dynamic way
String theType = GameConstants.ARCHER;

Map<String, int[]> allStats = new HashMap<>();
allStats.put(GameConstants.ARCHER, archerStats);
allStats.put("bomb", bombStats);
// Configure at class level
StandardUnit.initializeStats(allStats);

// Note - the type is 'unknown' here
Unit unit = new StandardUnit(theType, Player.GREEN);
dumpStats(unit);
```



Augment StandardUnit:

Class variable!

```
private static Map<String, int[]> allStats;
public static void initializeStats(Map<String, int[]> allStats) {
  StandardUnit.allStats = allStats;
}

public StandardUnit(String type, Player owner) {
  this.type = type;
  this.owner = owner;
  this.stats = allStats.get(type);
}
```

AARHUS UNIVERSITET

Morale

Use code for algorithms!

Use data structures for data!