AARHUS UNIVERSITET

Software Engineering and Architecture

Multi Dimensional Variance

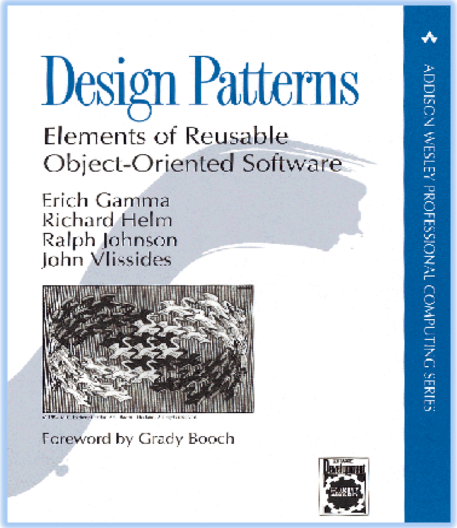
Ultra flexible software

Goal and means to an end?

- Patterns:
  - *Goal in itself or just the means to an end?*

- Patterns are interesting as *means* to achieve some specific quality in our software:
  - elements of **Reusable** ...

- A key aspect is handling **variance**


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

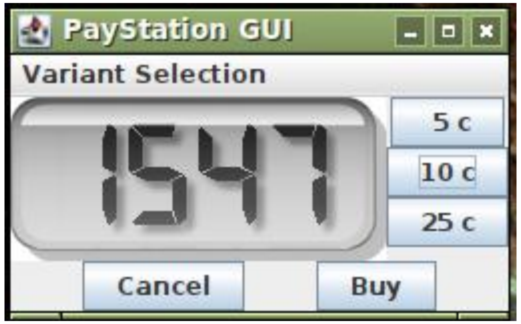
AARHUS UNIVERSITET

Variance

- Factoring out in roles and delegating to objects that play roles is a very strong technique to handle **multiple dimensions of variance!**
  - that is – a piece of software that must handle different types of context
    - work on both MariaDB and MongoDB database
    - work in both testing and production environment
    - work both with real hardware attached or simulated environment
    - work with variations for four different customers
- *Here all types of combinations are viable !*

New Requirements

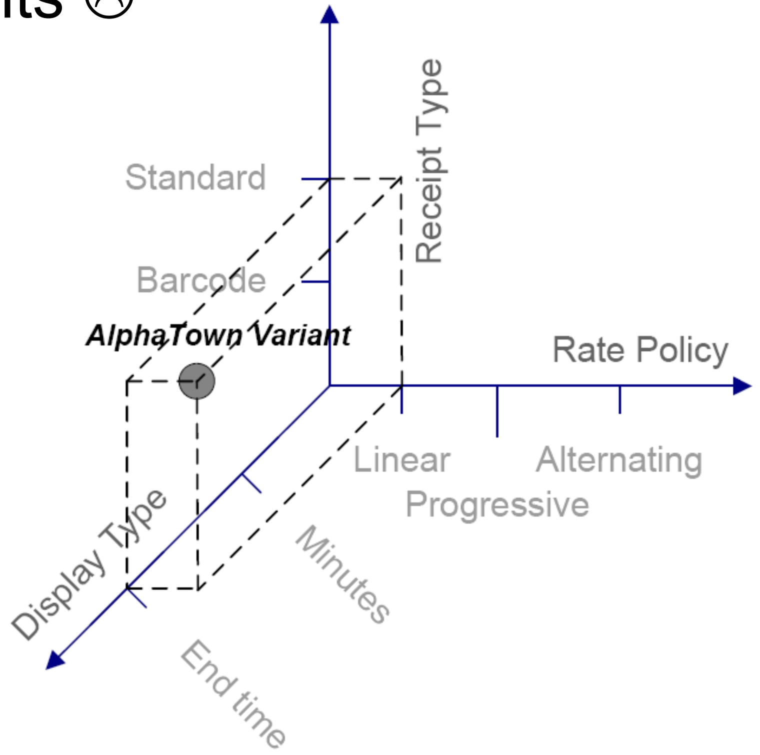
- Alphatown county wants the display to show *parking end time* instead of minutes bought!
  - I.e “Parking ends at 15:47”



AARHUS UNIVERSITET

Combinatorial explosion!

- All these requirements pose a *combinatorial explosion* of variants ☹



There are 3*2*2 = 12 combinations. This may be doubled if we include overriding weekend day algorithm !

AARHUS UNIVERSITET

Restating the Options

Parametric Variance

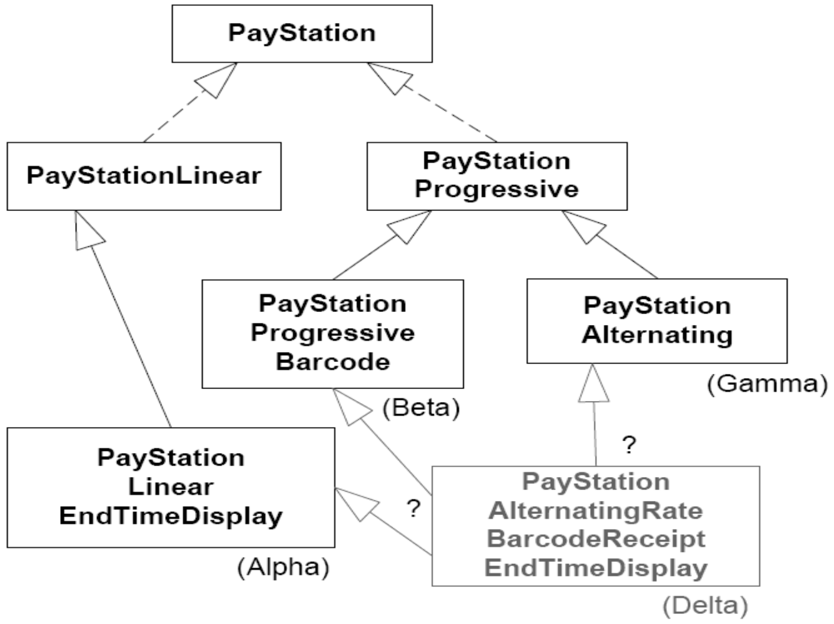
- Variant handling by **if (param**) or **#ifdef's** is well known, but the code simply bloats with conditional statements.

- Example: GNU C compiler has a single statement that includes 41 macro expansions !!!
- I wonder what that code does???

- #ifdef (MSDOS && ORACLE || MYSQL && …)
- #ifdef (DEBUG)
  - quickly you loose control of what is going on…

Polymorphic Variance

- Inheritance dies **miserably** facing this challenge!
- Just look at names!
- Making new variants is difficult.
- And code reuse is very difficult ☹


PayStation
PayStationLinear
PayStation Progressive
PayStation Progressive Barcode
(Beta)
PayStation Alternating
(Gamma)
PayStation Linear EndTimeDisplay
(Alpha)
?
?
PayStation AlternatingRate BarcodeReceipt EndTimeDisplay
(Delta)

Masking the problem

- By **combining** parametric and polymorphic variance you may mask the problem somewhat…
- I.e. handle receipt type by inheritance, and the rest by pumping the code with if's…
- but … it is still an inferior way to handle multi-dimensional variance…

Compositional software

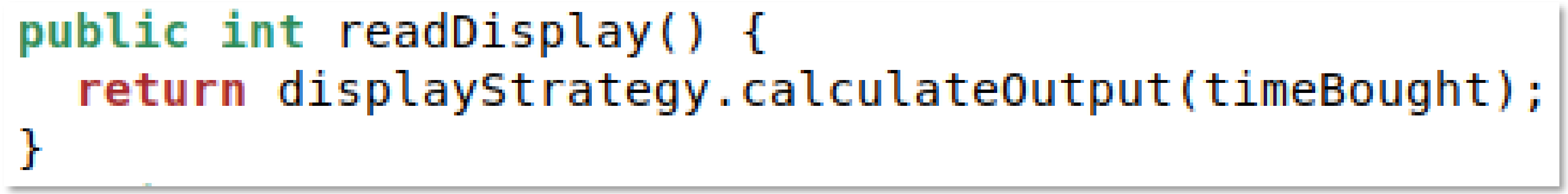
- The way forward is:
  - ***Compositional software***
  - Highly configurable and flexible software!

- ③ Consider what behavior that may vary
- ① Express variable behavior as a responsibility clearly defined by an interface
- ② Delegate to object serving the responsibility to perform behavior

AARHUS UNIVERSITET

③ ① ②

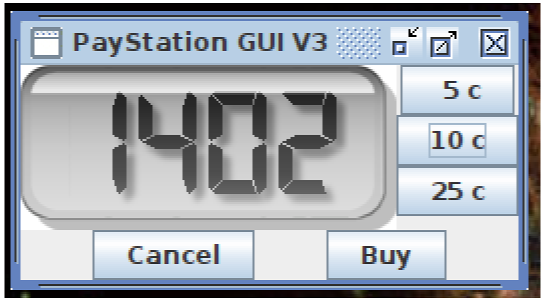
- ③ Encapsulate what varies
  - The display output must exist in variants
- ① Program to an interface
  - <<interface>> **DisplayStrategy**
    - public int calculateOutput(int minutes);
- ② Favor object composition

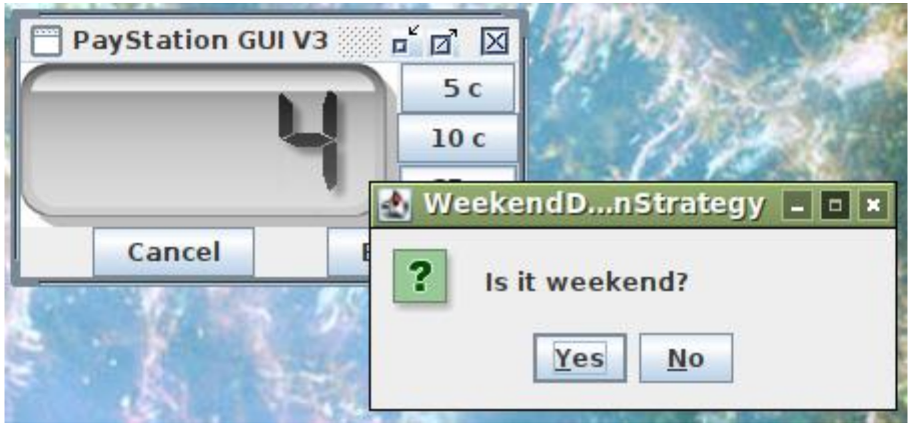


```
public int readDisplay() {
  return displayStrategy.calculateOutput(timeBought);
}
```

Compositional software

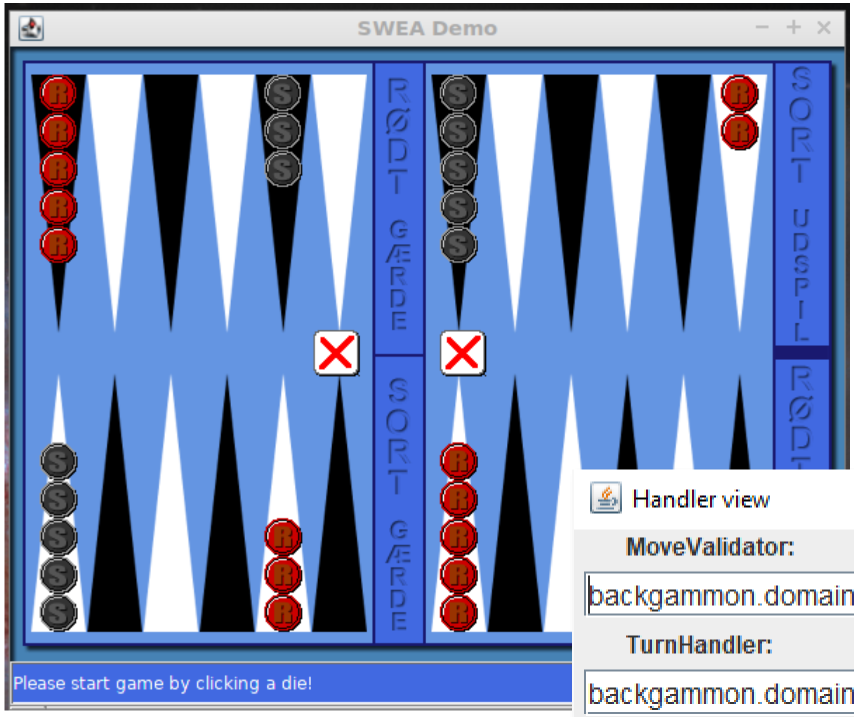
- [Demo]



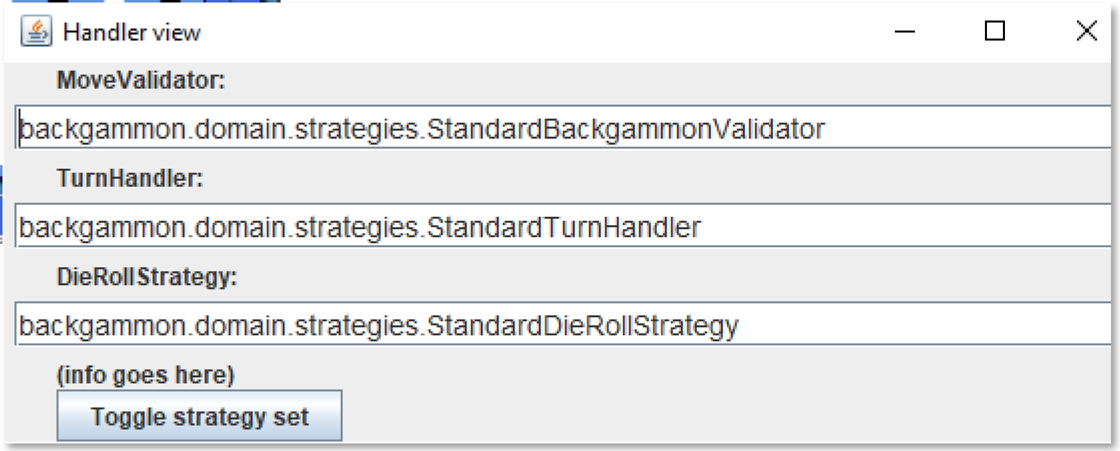


Is it weekend?

Compositional Software



Please start game by clicking a die!



MoveValidator:

backgammon.domain.strategies.StandardBackgammonValidator

TurnHandler:

backgammon.domain.strategies.StandardTurnHandler

DieRollStrategy:

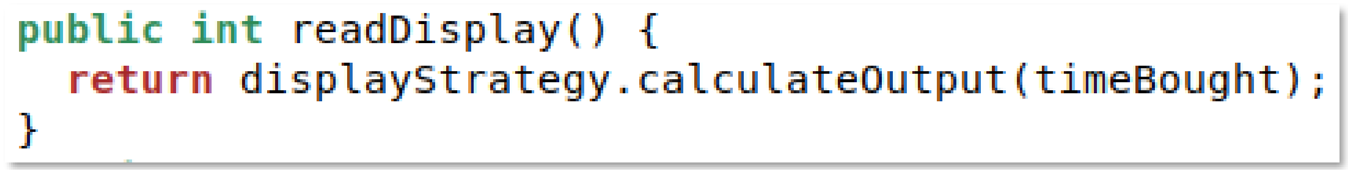
backgammon.domain.strategies.StandardDieRollStrategy

(info goes here)

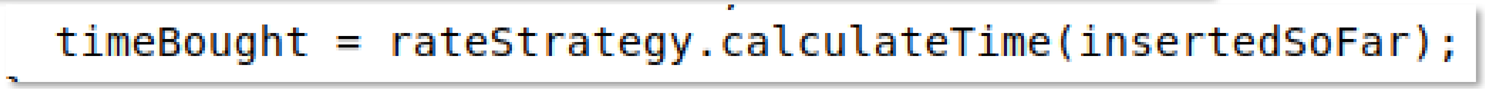
AARHUS UNIVERSITET

Compositional Software

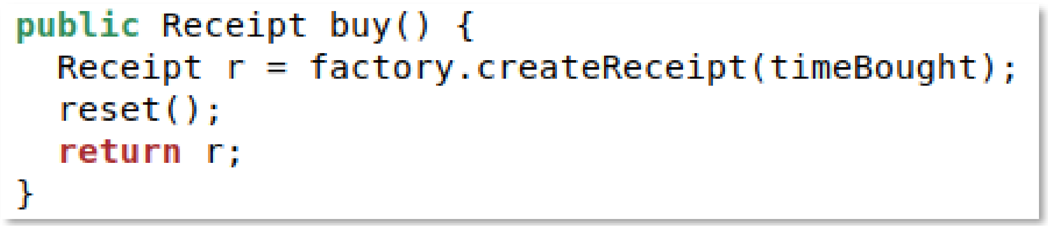
- The paystation has become a *team leader*, delegating jobs to specialist workers:



```
public int readDisplay() {
  return displayStrategy.calculateOutput(timeBought);
}
```



```
  timeBought = rateStrategy.calculateTime(insertedSoFar);
```



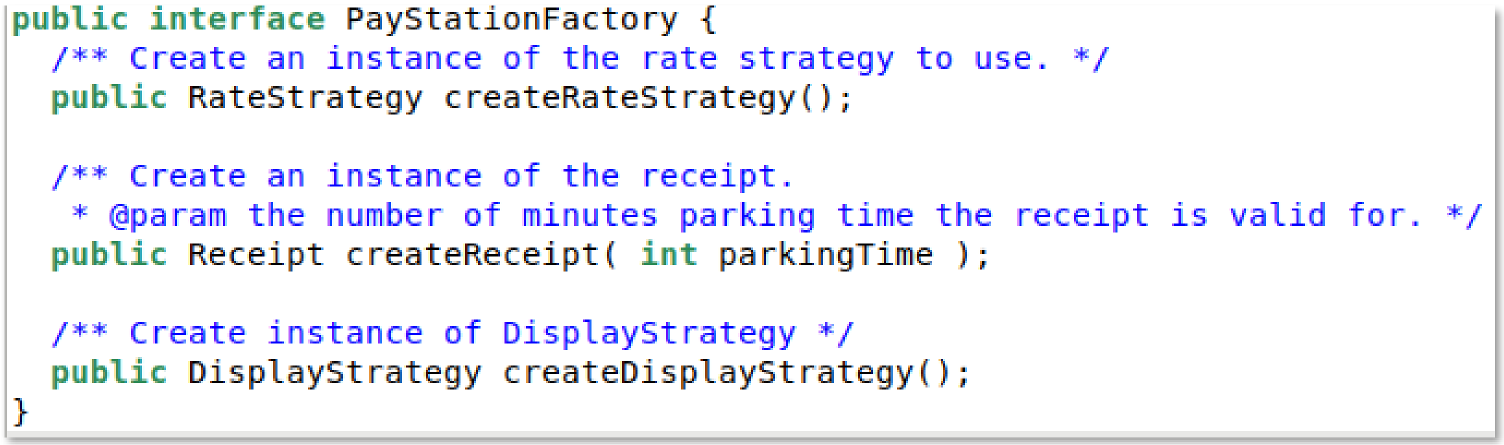
```
public Receipt buy() {
  Receipt r = factory.createReceipt(timeBought);
  reset();
  return r;
}
```

- Note! No if’s – no bloat – easy to read code leading to fewer bugs!

Compositional Software

- Telling the team leader which persons will serve the roles:

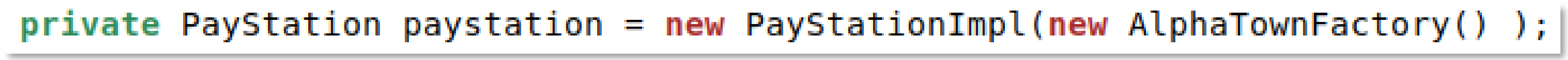
- The factory interface



```java
public interface PayStationFactory {
  /** Create an instance of the rate strategy to use. */
  public RateStrategy createRateStrategy();

  /** Create an instance of the receipt.
   * @param the number of minutes parking time the receipt is valid for. */
  public Receipt createReceipt( int parkingTime );

  /** Create instance of DisplayStrategy */
  public DisplayStrategy createDisplayStrategy();
}
```

Compositional Software

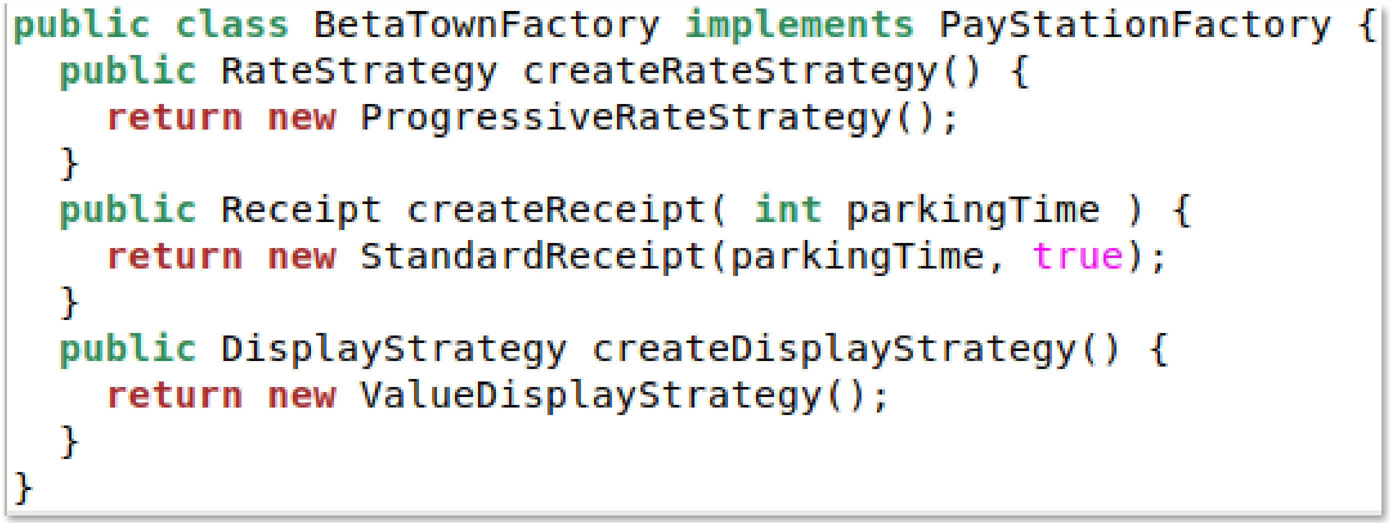
- Creating a pay station:
  - create the factory
  - create the pay station, giving it access to the factory



```java
private PayStation paystation = new PayStationImpl(new AlphaTownFactory() );
```

Compositional Software

- ... and a factory:



```java
public class BetaTownFactory implements PayStationFactory {
  public RateStrategy createRateStrategy() {
    return new ProgressiveRateStrategy();
  }
  public Receipt createReceipt( int parkingTime ) {
    return new StandardReceipt(parkingTime, true);
  }
  public DisplayStrategy createDisplayStrategy() {
    return new ValueDisplayStrategy();
  }
}
```

AARHUS UNIVERSITET

Analysis

- Benefits
  - The variability points are independent
    - we introduced new display strategy – but this did not alter any of the existing strategies !

```
public int readDisplay() {
  return displayStrategy.calculateOutput(timeBought);
}
```

  - Once the variability point has been introduced we can introduce as many new types of variations as we like – only by *adding* new classes
    - any price model; new receipt types; new display output...
  - **Open-closed principle in action...**

Open/Closed principle

Open for extension

Closed for modification

AARHUS UNIVERSITET

Open/Closed principle

- **Open** for extension
  - I can make my own feature additions/changes by extending the software
- **Closed** for modification
  - But I do not rewrite any existing code
    - Or ask Oracle, Google, NetFlix, Apache, to rewrite code to handle my extensions

- (i.e. no soldering of wires in my TV set)

AARHUS UNIVERSITET

Analysis

- Benefits
  - Any combination you want, we are able to “mix”
  - Nonsense combinations can be delimited
    - abstract factory is the place to “mix” the cocktails
  - Code readability
    - every aspect of the configuration is clearly defined in a single place
      - configuration mixing in the abstract factory
      - orchestration in the PayStation impl
      - each variation type in its own implementing class

AARHUS UNIVERSITET

Analysis

- Liabilities
  - Each dimension of variability (price model, receipt type, display output, etc) is *really* independent – so

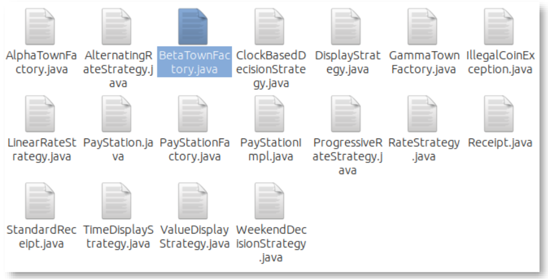
  - we cannot feed information from one to the other directly ☹

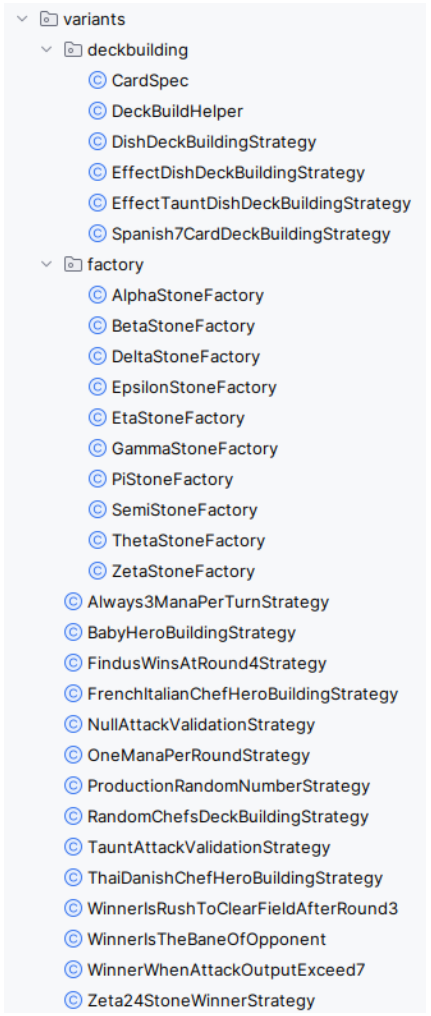
  - If they require information from each other
    - Then of course we must provide the means to do so
      - Mediator pattern, memento pattern, observer pattern, others
    - Like we do in mandatory project
      - StandardGame calls strategy with ‘this’ ala a Private Interface
      - The Strategy calls mutators on the MutableGame

Analysis

- Liabilities
- The number of classes in action ☹

- On the other hand:
  - careful naming makes it possible to quickly identify which class to change…
- And use packages to group cohesive modules


AlphaTownFactory.java AlternatingRateStrategy.java BetaTownFactory.java ClockBasedDecisionStrategy.java DisplayStrategy.java GammaTownFactory.java IllegalCoinException.java
LinearRateStrategy.java PayStation.java PayStationFactory.java PayStationImpl.java ProgressiveRateStrategy.java RateStrategy.java Receipt.java
StandardReceipt.java TimeDisplayStrategy.java ValueDisplayStrategy.java WeekendDecisionStrategy.java


```
variants
  deckbuilding
    CardSpec
    DeckBuildHelper
    DishDeckBuildingStrategy
    EffectDishDeckBuildingStrategy
    EffectTauntDishDeckBuildingStrategy
    Spanish7CardDeckBuildingStrategy
  factory
    AlphaStoneFactory
    BetaStoneFactory
    DeltaStoneFactory
    EpsilonStoneFactory
    EtaStoneFactory
    GammaStoneFactory
    PiStoneFactory
    SemiStoneFactory
    ThetaStoneFactory
    ZetaStoneFactory
  Always3ManaPerTurnStrategy
  BabyHeroBuildingStrategy
  FindusWinsAtRound4Strategy
  FrenchItalianChefHeroBuildingStrategy
  NullAttackValidationStrategy
  OneManaPerRoundStrategy
  ProductionRandomNumberStrategy
  RandomChefsDeckBuildingStrategy
  TauntAttackValidationStrategy
  ThaiDanishChefHeroBuildingStrategy
  WinnerIsRushToClearFieldAfterRound3
  WinnerIsTheBaneOfOpponent
  WinnerWhenAttackOutputExceed7
  Zeta24StoneWinnerStrategy
```

Analysis

- Liabilities
  - Actually I have a combinatorial explosion of factories! I need a factory for each and every combination of delegates that I have
  - Exercise: How can I avoid this explosion?

AARHUS UNIVERSITET

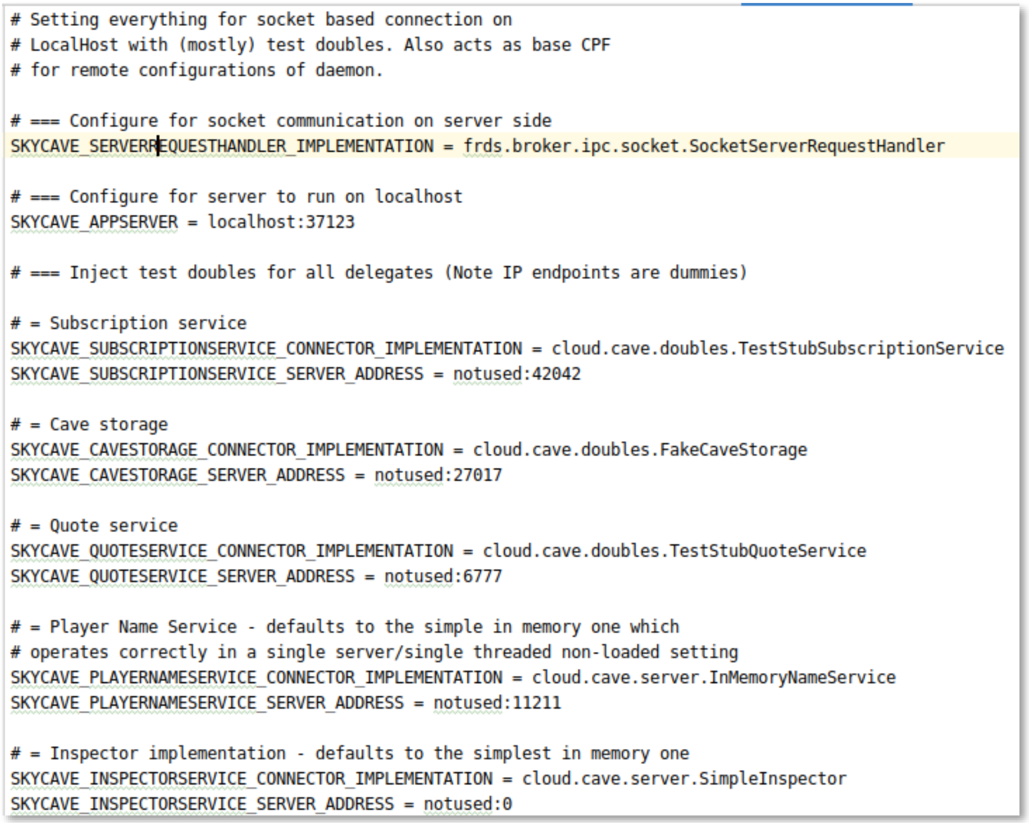
Another Example

SkyCave

AARHUS UNIVERSITET

Configuration System

- Six roles of variability
  - Storage system
  - Network connector
  - Authentication
  - External services
  - Name Service
  - Logging System
- AbsFactory reads a *CPF property file*
  - *Impl class*
  - *Network host and port*

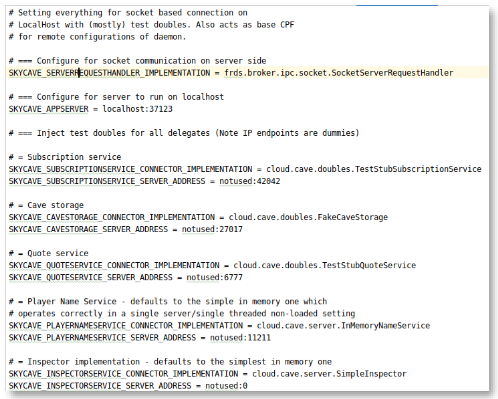


```
# Setting everything for socket based connection on
# LocalHost with (mostly) test doubles. Also acts as base CPF
# for remote configurations of daemon.

# === Configure for socket communication on server side
SKYCAVE_SERVERREQUESTHANDLER_IMPLEMENTATION = frds.broker.ipc.socket.SocketServerRequestHandler

# === Configure for server to run on localhost
SKYCAVE_APPSERVER = localhost:37123

# === Inject test doubles for all delegates (Note IP endpoints are dummies)

# = Subscription service
SKYCAVE_SUBSCRIPTIONSERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.TestStubSubscriptionService
SKYCAVE_SUBSCRIPTIONSERVICE_SERVER_ADDRESS = notused:42042

# = Cave storage
SKYCAVE_CAVESTORAGE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.FakeCaveStorage
SKYCAVE_CAVESTORAGE_SERVER_ADDRESS = notused:27017

# = Quote service
SKYCAVE_QUOTESERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.TestStubQuoteService
SKYCAVE_QUOTESERVICE_SERVER_ADDRESS = notused:6777

# = Player Name Service - defaults to the simple in memory one which
# operates correctly in a single server/single threaded non-loaded setting
SKYCAVE_PLAYERNAMESERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.server.InMemoryNameService
SKYCAVE_PLAYERNAMESERVICE_SERVER_ADDRESS = notused:11211

# = Inspector implementation - defaults to the simplest in memory one
SKYCAVE_INSPECTORSERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.server.SimpleInspector
SKYCAVE_INSPECTORSERVICE_SERVER_ADDRESS = notused:0
```

AARHUS UNIVERSITET

Configuration System

- Six roles of variability
  - Storage system (5)
    - FakeObject, MongoDB, Redis, Memcached, MariaDB
  - Network connector (3)
    - Sockets, HTTP, RabbitMQ
  - Authentication (3)
    - TestStub, NullObject, RealService
  - External services (2)
    - TestStub, RealService
  - Name Service (2)
    - In memory, Memcached
  - Logging System (2)
    - In memory, Memcached

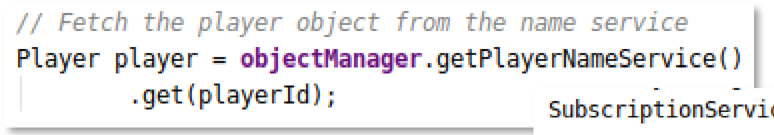


```
# Setting everything for socket based connection on
# LocalHost with (mostly) test doubles. Also acts as base CPF
# for remote configurations of daemon.

# === Configure for socket communication on server side
SKYCAVE_SERVERREQUESTHANDLER_IMPLEMENTATION = frds.broker.ipc.socket.SocketServerRequestHandler

# === Configure for server to run on localhost
SKYCAVE_APPSERVER = localhost:37123

# === Inject test doubles for all delegates (Note IP endpoints are dummies)

# = Subscription service
SKYCAVE_SUBSCRIPTIONSERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.TestStubSubscriptionService
SKYCAVE_SUBSCRIPTIONSERVICE_SERVER_ADDRESS = notused:42042

# = Cave storage
SKYCAVE_CAVESTORAGE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.FakeCaveStorage
SKYCAVE_CAVESTORAGE_SERVER_ADDRESS = notused:27017

# = Quote service
SKYCAVE_QUOTESERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.doubles.TestStubQuoteService
SKYCAVE_QUOTESERVICE_SERVER_ADDRESS = notused:6777

# = Player Name Service - defaults to the simple in memory one which
# operates correctly in a single server/single threaded non-loaded setting
SKYCAVE_PLAYERNAMESERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.server.InMemoryNameService
SKYCAVE_PLAYERNAMESERVICE_SERVER_ADDRESS = notused:11211

# = Inspector implementation - defaults to the simplest in memory one
SKYCAVE_INSPECTORSERVICE_CONNECTOR_IMPLEMENTATION = cloud.cave.server.SimpleInspector
SKYCAVE_INSPECTORSERVICE_SERVER_ADDRESS = notused:0
```

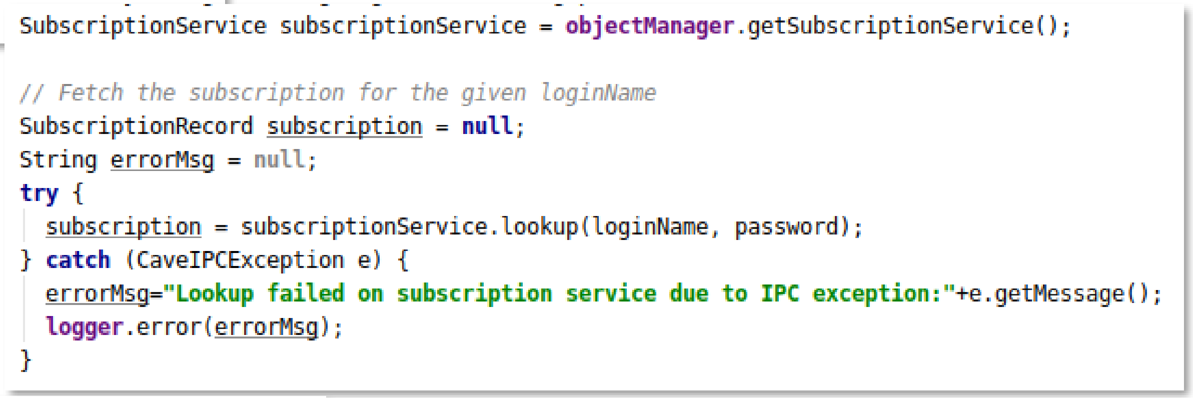
SkyCave can exist in 5*3*3*2*2*2

= 360 variants

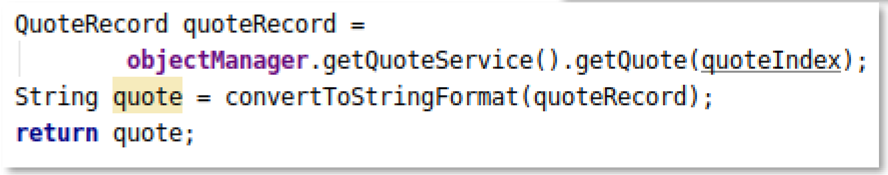
And No Code Clutter



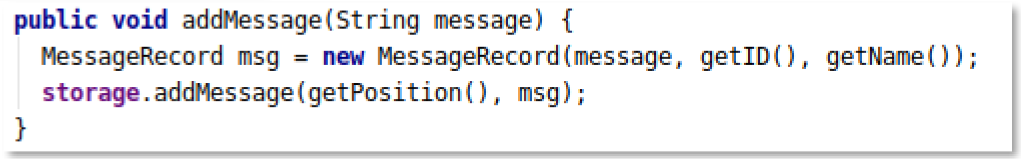
```
// Fetch the player object from the name service
Player player = objectManager.getPlayerNameService()
        .get(playerId);
```



```
SubscriptionService subscriptionService = objectManager.getSubscriptionService();

// Fetch the subscription for the given loginName
SubscriptionRecord subscription = null;
String errorMsg = null;
try {
  subscription = subscriptionService.lookup(loginName, password);
} catch (CaveIPCException e) {
  errorMsg="Lookup failed on subscription service due to IPC exception:"+e.getMessage();
  logger.error(errorMsg);
}
```



```
QuoteRecord quoteRecord =
        objectManager.getQuoteService().getQuote(quoteIndex);
String quote = convertToStringFormat(quoteRecord);
return quote;
```



```
public void addMessage(String message) {
  MessageRecord msg = new MessageRecord(message, getID(), getName());
  storage.addMessage(getPosition(), msg);
}
```

- *An object manager keeps track of all delegates ☺*

Summary

Handle multi-dimensional variance by compositional software designs !