Designing Classes

CPSC 1181 – O.O.

Jeremy Hilliker

Summer 2017

Langara.
THE COLLEGE OF HIGHER LEARNING.

Overview

- Discovering classes
- Cohesion
- Dependency
- Coupling
- Side Effects

Discovering Classes

- **Nouns**
  - Classes?
    - First Class Citizen?
  - Composite attribute?
  - Represents a single concept from the domain.
  - "Is a" relationship
- Verbs
  - Behaviours (mutators)
- Questions
  - Accessors, Predicates
- Adjectives
  - Attributes ("Has a" relationship)
  - Parameters

- You are to implement a class called ShoppingCart. It will be a virtual representation of all of the things that you want to buy. But there are a few catches! You only have so much money to spend, and you can only carry so much weight.
- When you create your shopping cart, you tell it your budget and weight limit. You can then start adding items to your cart. Each item will have a name (like: "high-waist plaid skater skirt" or "Fender Telecaster"), a before tax price (like: $24.99 or $800.00), and a weight (like: 300g or 3600g). Values on an item cannot be changed. Use a float to represent price.
- At any time, you can ask your shopping cart for the total weight of its contents, if it is overweight, and how much more weight you can still add to it before it becomes overweight. You can also ask your cart to tell you what item within it is the heaviest item.
- Likewise for your budget and price, except it will give you the after tax figures (assume that tax is 12% on all items, and that you are not tax exempt).
- Sometimes you forget if you have put something in your cart, so you can ask your cart if it holds an item by name. You may also want to see the details of an item, so you likewise ask it for the item by name -- your cart returns a fitted array containing all items with that name.
- Sometimes you change your mind about what you want to buy and you want to remove an item. You can do this by giving the cart the item's name. If there is more than one item with that name, your cart will remove any one item of that name. You can also remove things from the cart by telling the cart the exact item to remove, in which case it will remove that one item. Or you can tell the cart to remove all items with a given name, in which case it will do so, and it will tell you how many items it removed.
- Sometimes you get frustrated and completely empty your cart.
- Finally, you can ask your cart to tell you all of the items that it contains. The list that it gives you must not be modifiable. You do not have to guarantee that this list stays synchronized with the contents of your cart, nor that it will be unchanged if the cart's contents change. The list is only valid for as long as the cart's contents do not change. There are no further guarantees of its behavior than that.

Nouns

- Obvious:
  - BankAccount
  - Paycheck
- Actors (end in –er or –or)
  - Do some kind of work
  - Scanner
  - RandomNumberGenerator
  - Services
- Don’t turn verbs into Classes (unless you’re really sure)
- Utility Classes
  - No objects
  - Only static methods
  - Math
- Program starters
  - main

Check

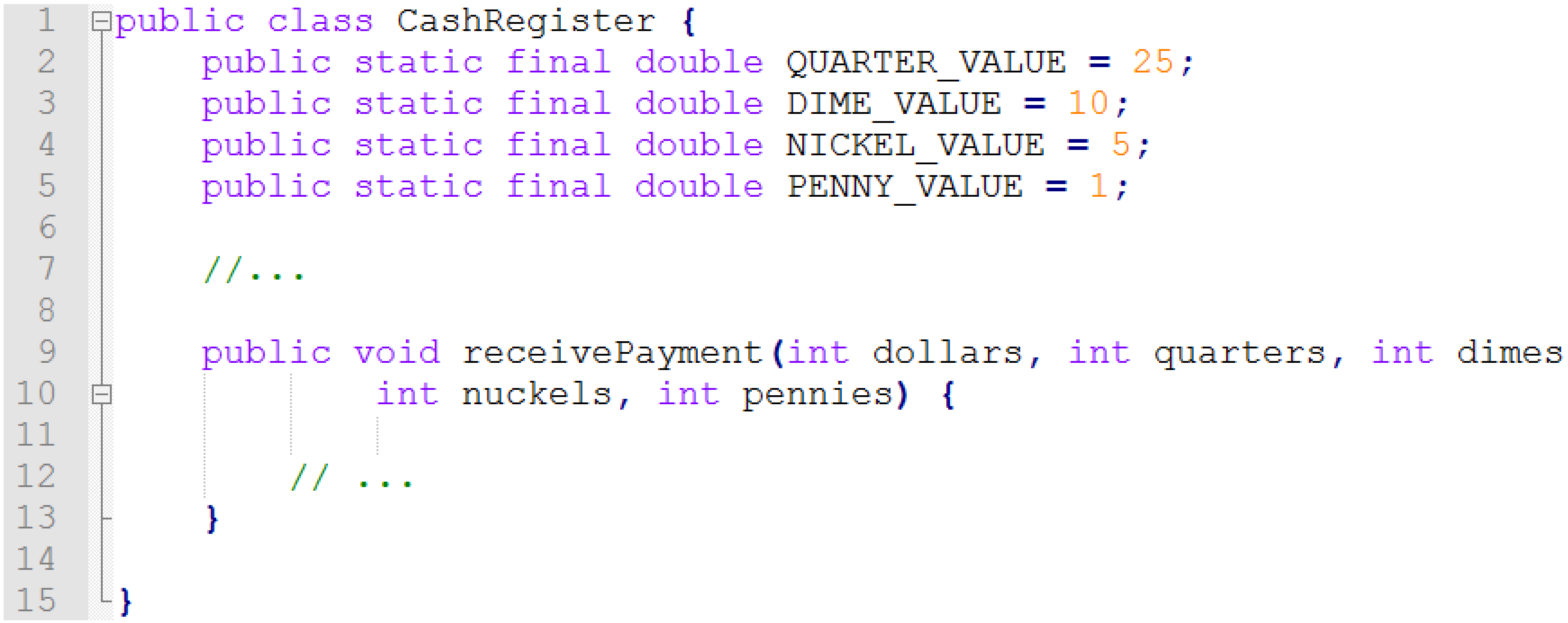
- We’re writing a chess program
- Are these potential classes:

ChessBoard	
Piece	
MovePiece	
RemovePiece	
CanMove	
PieceMover	
Referee	
BoardManager	

Cohesion

- The degree to which the elements of a module work together.
  - The strength of relationship between elements within a module
- Indicator of: class represents a single concept
- How to smell: is there coarse grained access to data from methods?

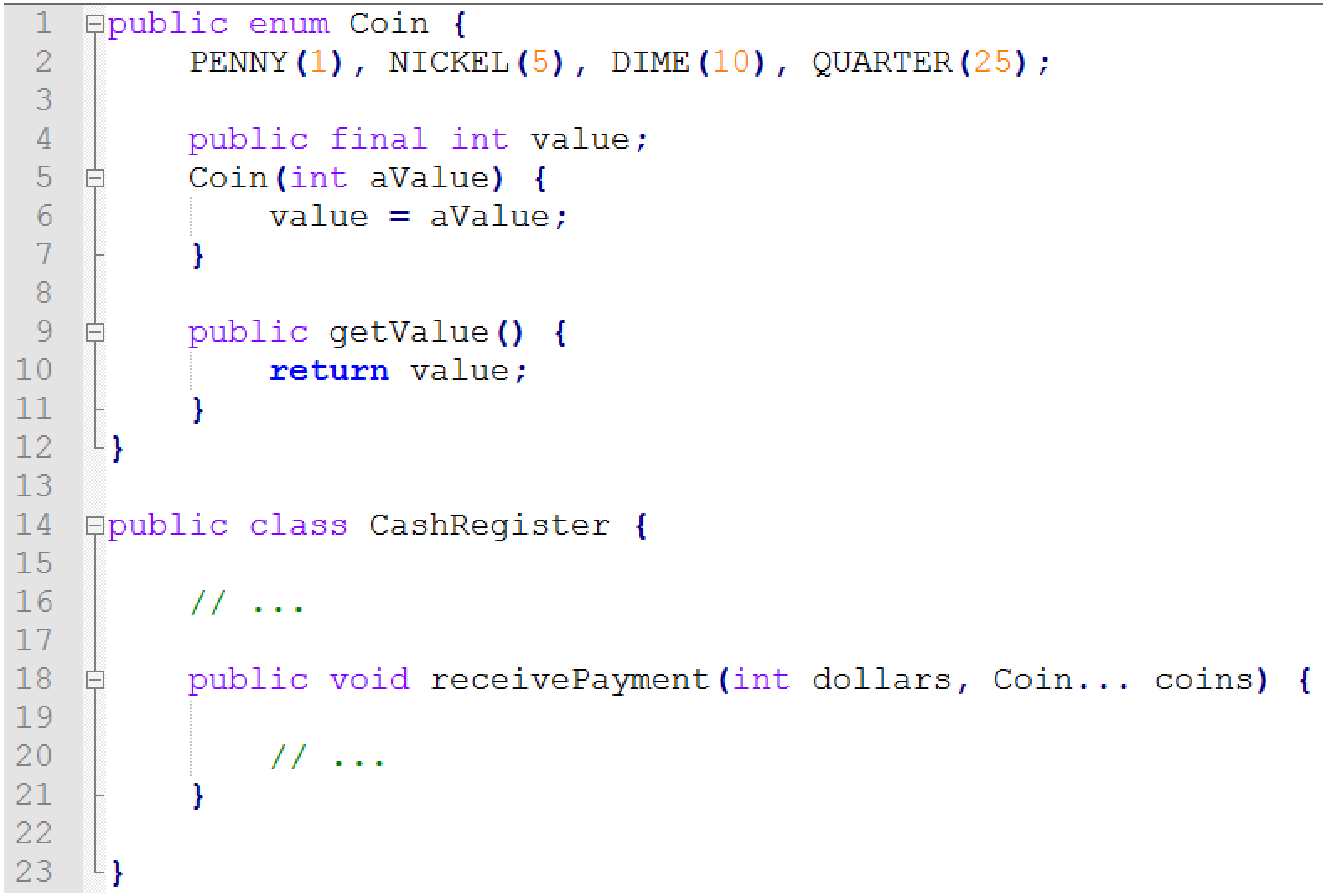
Cohesion – Ex (poor)



```java
public class CashRegister {
    public static final double QUARTER_VALUE = 25;
    public static final double DIME_VALUE = 10;
    public static final double NICKEL_VALUE = 5;
    public static final double PENNY_VALUE = 1;

    //...

    public void receivePayment(int dollars, int quarters, int dimes
                int nuckels, int pennies) {

        // ...
    }

}
```

- Two concepts
  - A cash register that does transactions, holds ***coins***, computes values
  - The values of individual ***coins***

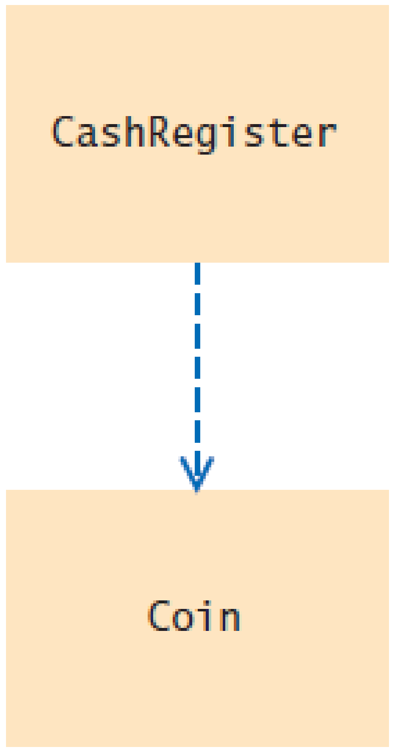
Cohesion – Ex (fixed)



```java
public enum Coin {
        PENNY(1), NICKEL(5), DIME(10), QUARTER(25);

        public final int value;
        Coin(int aValue) {
                value = aValue;
        }

        public getValue() {
                return value;
        }
}

public class CashRegister {

        // ...

        public void receivePayment(int dollars, Coin... coins) {

                // ...
        }

}
```

Minimizing Dependency

- A class depends on another class if it uses that class in any way
  - CashRegister depends on Coin



Check

- How does
  - CashRegister depend on Coin
    - Passed as argument: “uses”
  - Coin depend on CashRegister
    - It doesn’t

- Why minimize dependencies between classes?
  - Limits reuse
  - Harder to test
  - Harder to use
  - A change in one effects the others (connascence)

Dependency ~ Coupling

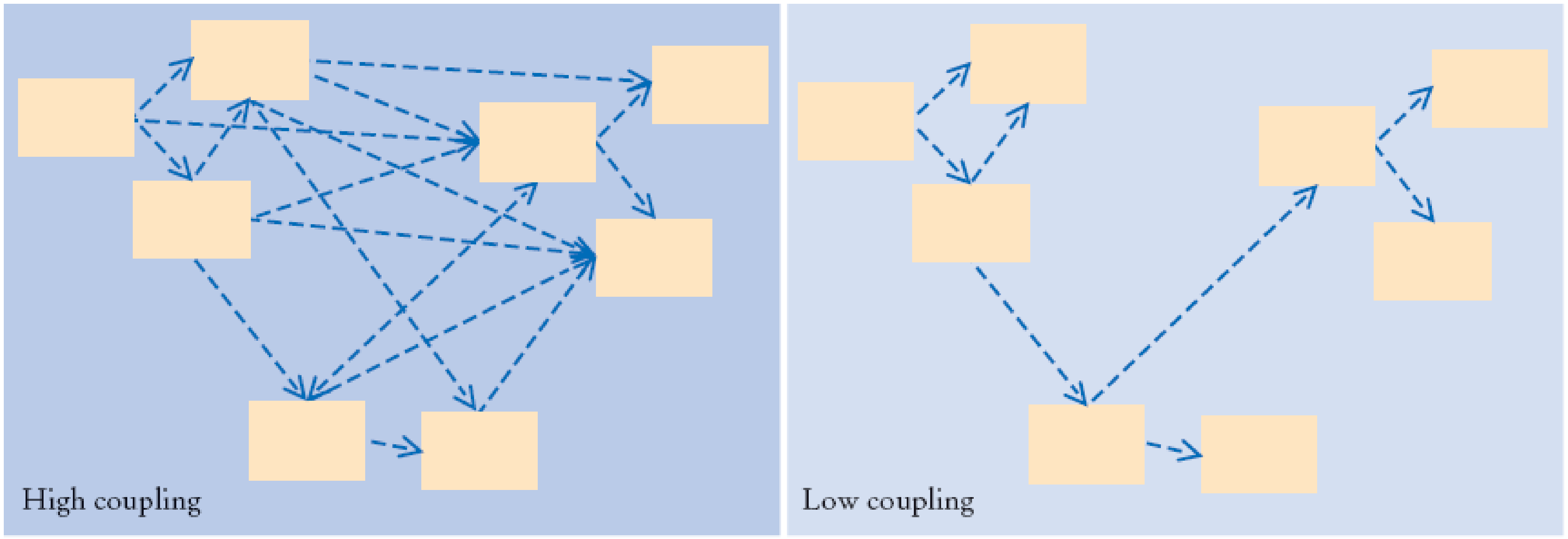


Figure 2 High and Low Coupling Between Classes

Minimizing Dependency Ex:

- Printing the balance
- Consider:

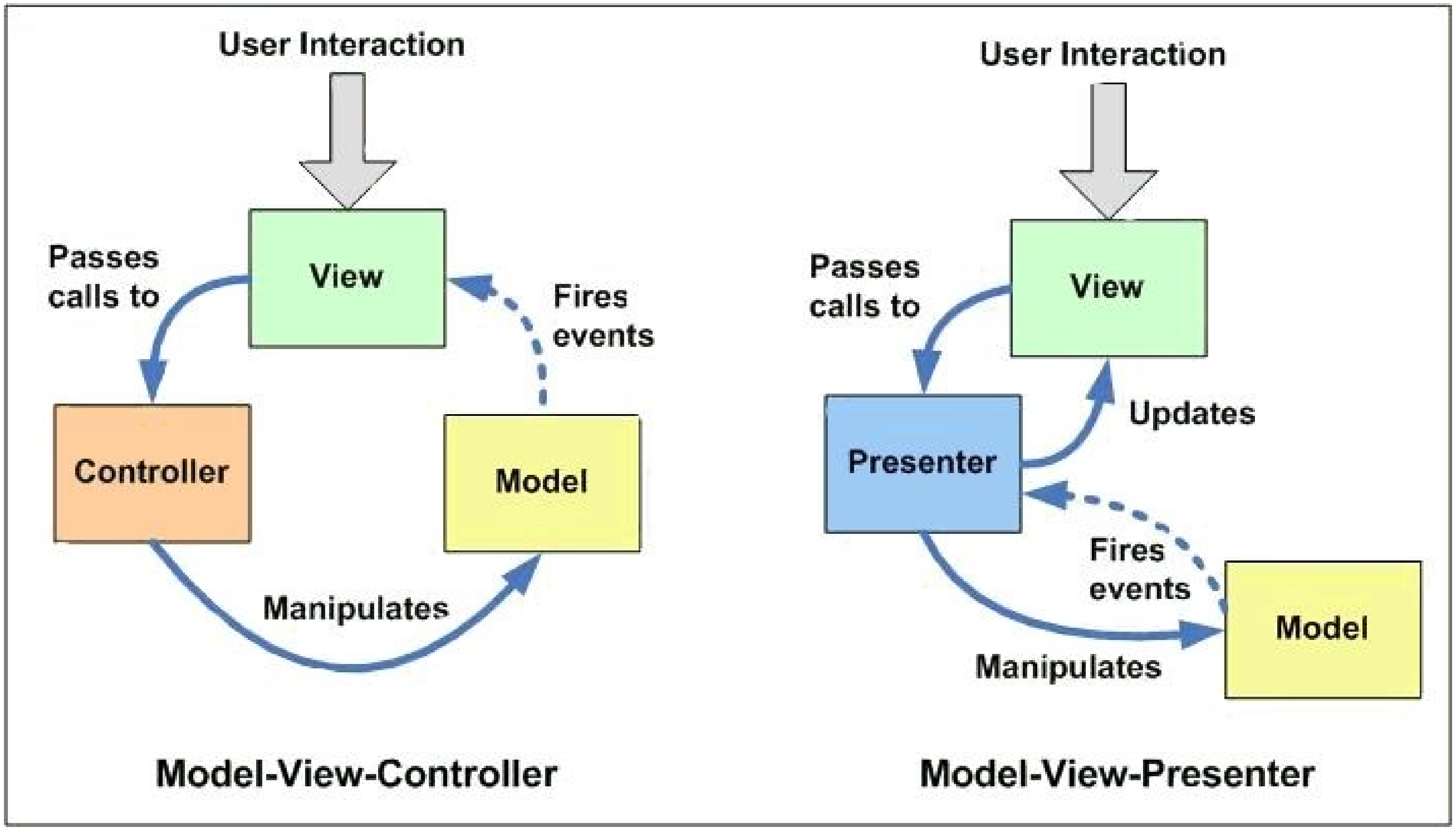
```
public void printBalance() {
    System.out.println(this.balance);
}
```

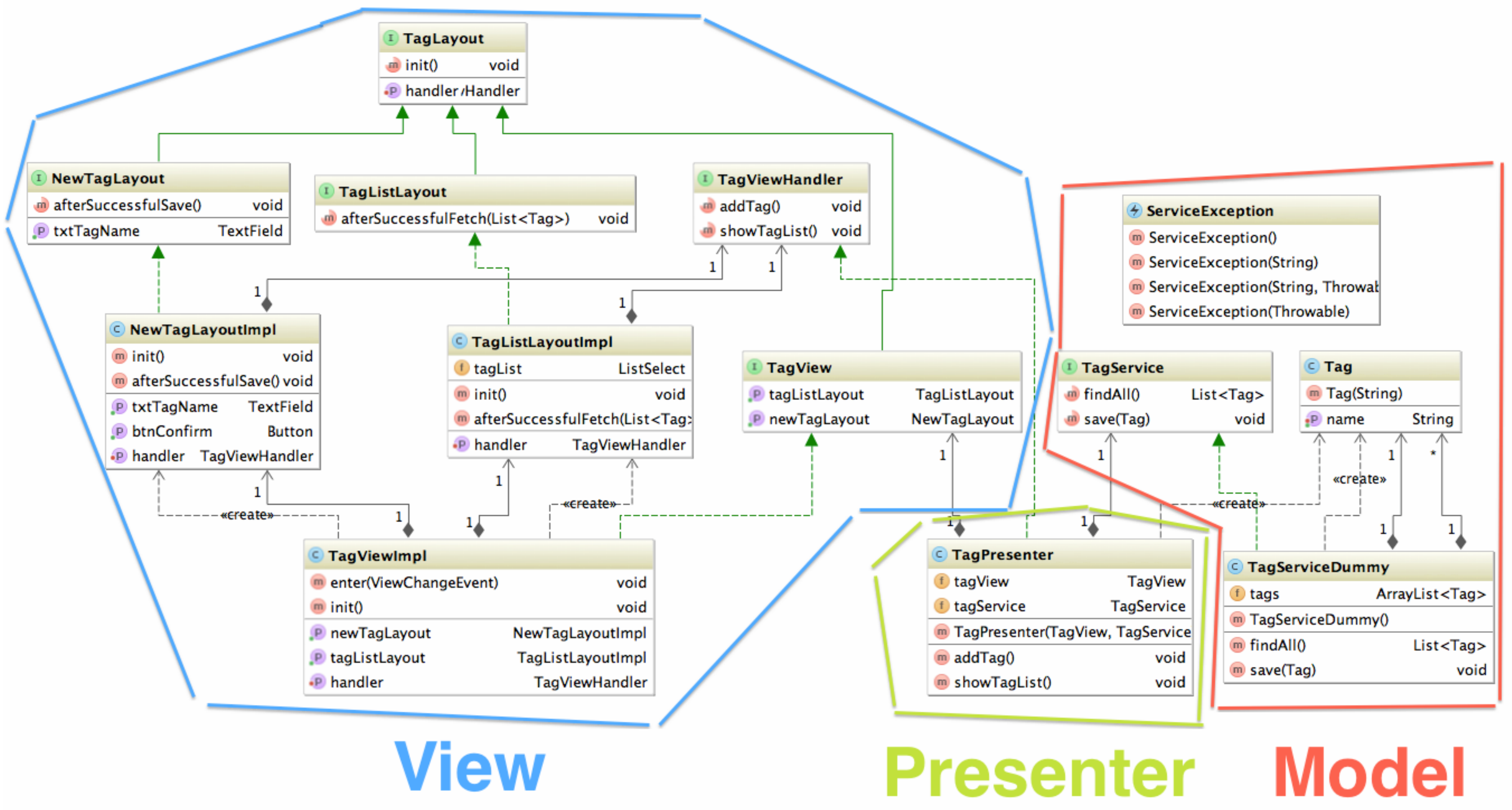
- Has a hard dependency on `System.out`

- Alternative:

```
System.out.println( cashRegister.getBalance() );
```

- Should decouple I/O from general work
  - Recall: MVC & MVP


User Interaction
View
Passes calls to
Fires events
Controller
Model
Manipulates
Model-View-Controller
User Interaction
View
Passes calls to
Presenter
Updates
Fires events
Model
Manipulates
Model-View-Presenter


TagLayout
init() void
handler Handler
NewTagLayout
afterSuccessfulSave() void
txtTagName TextField
TagListLayout
afterSuccessfulFetch(List<Tag>) void
TagViewHandler
addTag() void
showTagList() void
NewTagLayoutImpl
init() void
afterSuccessfulSave() void
txtTagName TextField
btnConfirm Button
handler TagViewHandler
TagListLayoutImpl
tagList ListSelect
init() void
afterSuccessfulFetch(List<Tag>
handler TagViewHandler
TagView
tagListLayout TagListLayout
newTagLayout NewTagLayout
TagViewImpl
enter(ViewChangeEvent) void
init() void
newTagLayout NewTagLayoutImpl
tagListLayout TagListLayoutImpl
handler TagViewHandler
ServiceException
ServiceException()
ServiceException(String)
ServiceException(String, Throwal
ServiceException(Throwable)
TagService
findAll() List<Tag>
save(Tag) void
Tag
Tag(String)
name String
TagPresenter
tagView TagView
tagService TagService
TagPresenter(TagView, TagService
addTag() void
showTagList() void
TagServiceDummy
tags ArrayList<Tag>
TagServiceDummy()
findAll() List<Tag>
save(Tag) void
«create»
1
*
View
Presenter
Model

Separating Accessors and Mutators

- Mutator
  - Changes state
  - Return type should be void
    - Can return an informational value
      - State changed?
      - How many removed
- Accessor
  - Asks a question
  - Can compute a value
  - Should not change state

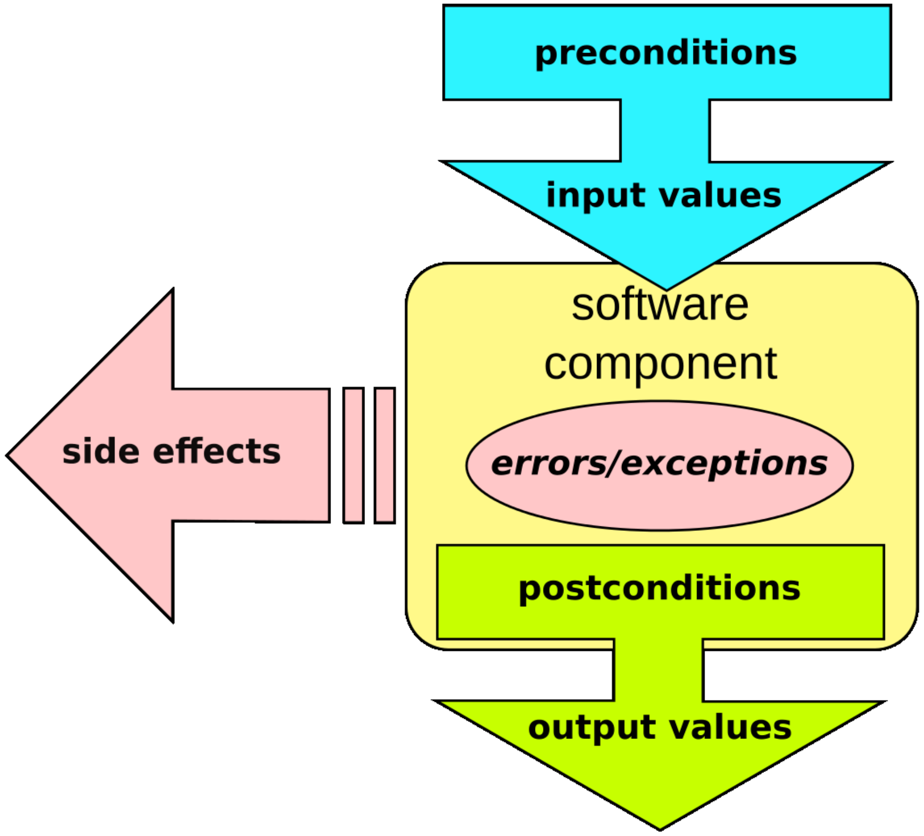
Check

- Is String.substring() a accessor or mutator?
  - Accessor. State does not change (String is immutable)
- Is Rectangle immutable? Why?
  - No: translate(x,y)

Side Effects

- A change in state
- Or an observable interaction with
  - The caller
  - Or the outside world

- All mutators have side effects
  - State change



Check

- Consider:

```
((BankAccount) a).deposit(100)
```

- Is it a mutator or accessor?
    - Mutator.
- If so, what is the side effect?
    - Changes balance in object a

Check

- Consider:

```
void read(Scanner in) {
   while(in.hasNextDouble())
         addScore(in.nextDouble())
}
```

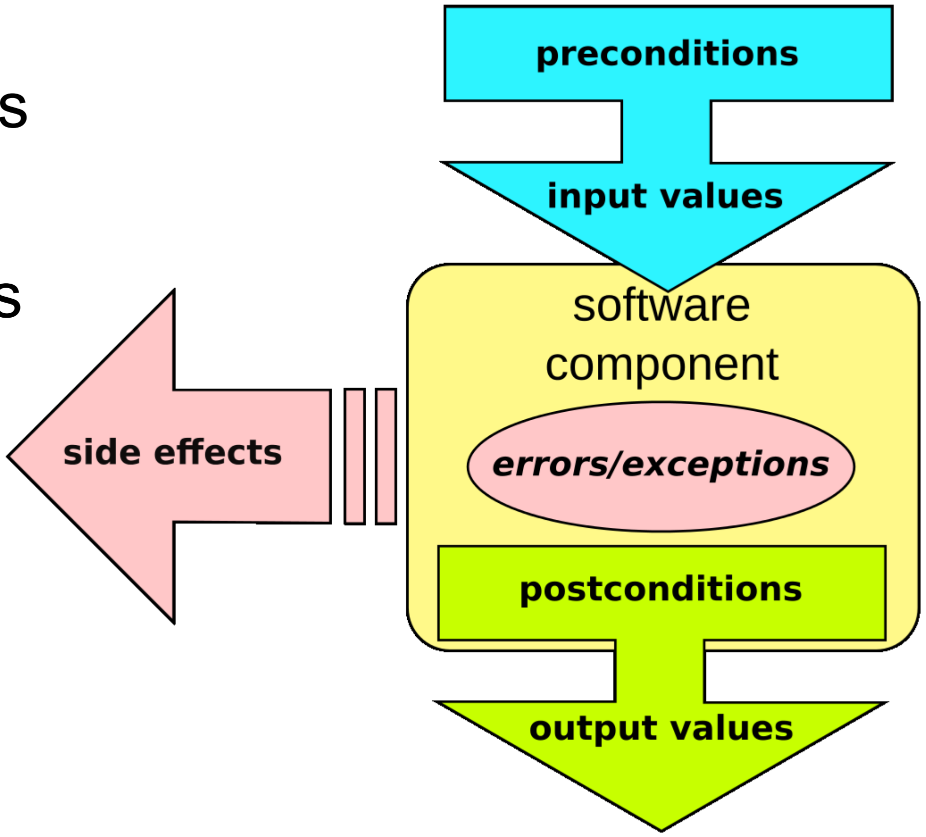
- What side effects does it have?

  - Mutates this through "`addScore`"
  - Mutates `Scanner in` through in.`nextDouble()`

- Is there an accessor?
  - Yes: `in.hasNextDouble()`

Avoiding Side Effects

- Goal:
  - minimize side effects
- Bad:
  - Mutating parameters
  - I/O in a class
- See:
  - Separation of Concerns



Recap

- Discovering classes
    - nouns, adjectives, verbs, questions
- Cohesion
    - Good
- Dependency
    - Avoid where possible
- Coupling
    - Bad
- Side Effects
    - Avoid where not required