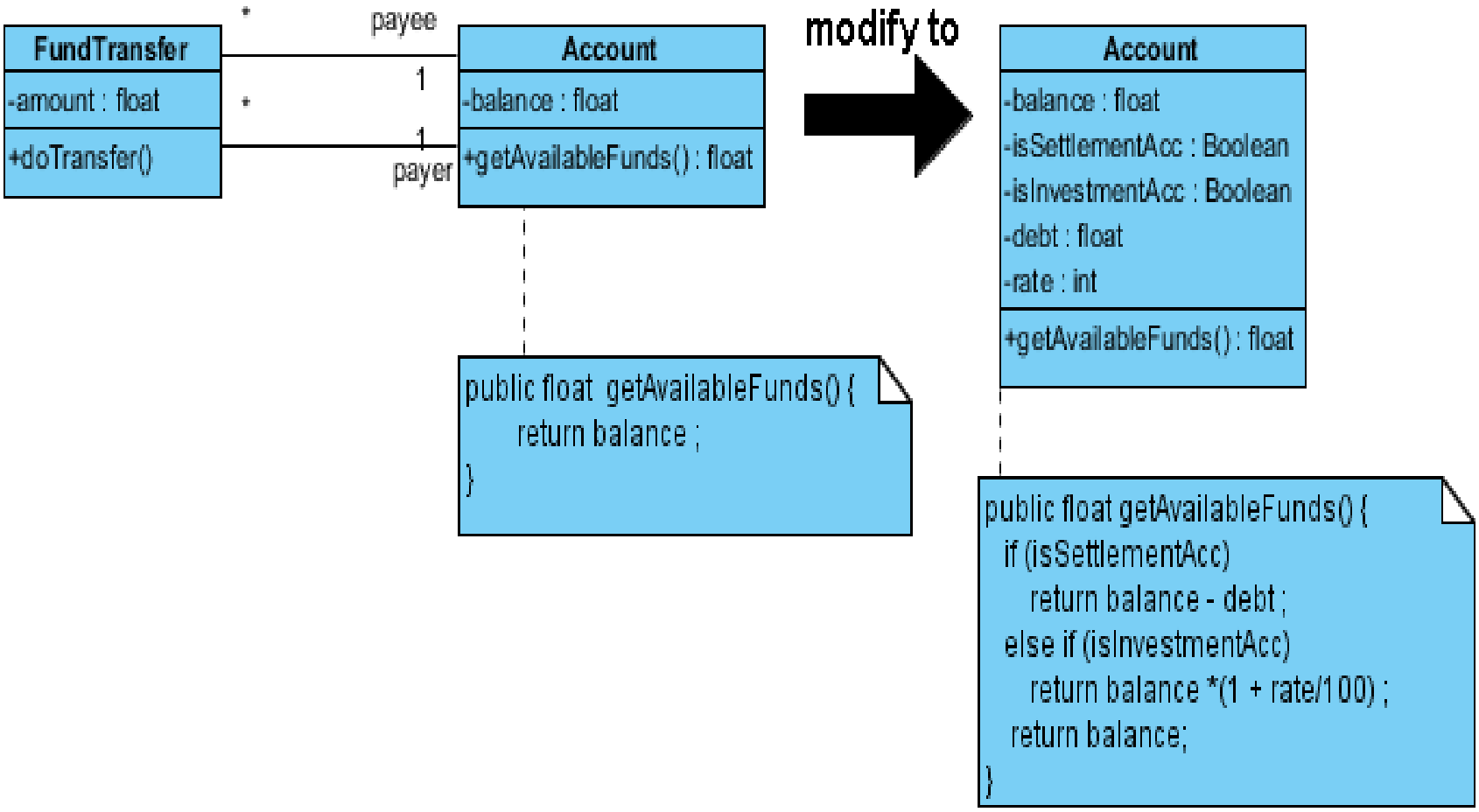
Tutorial 7

Design Principles

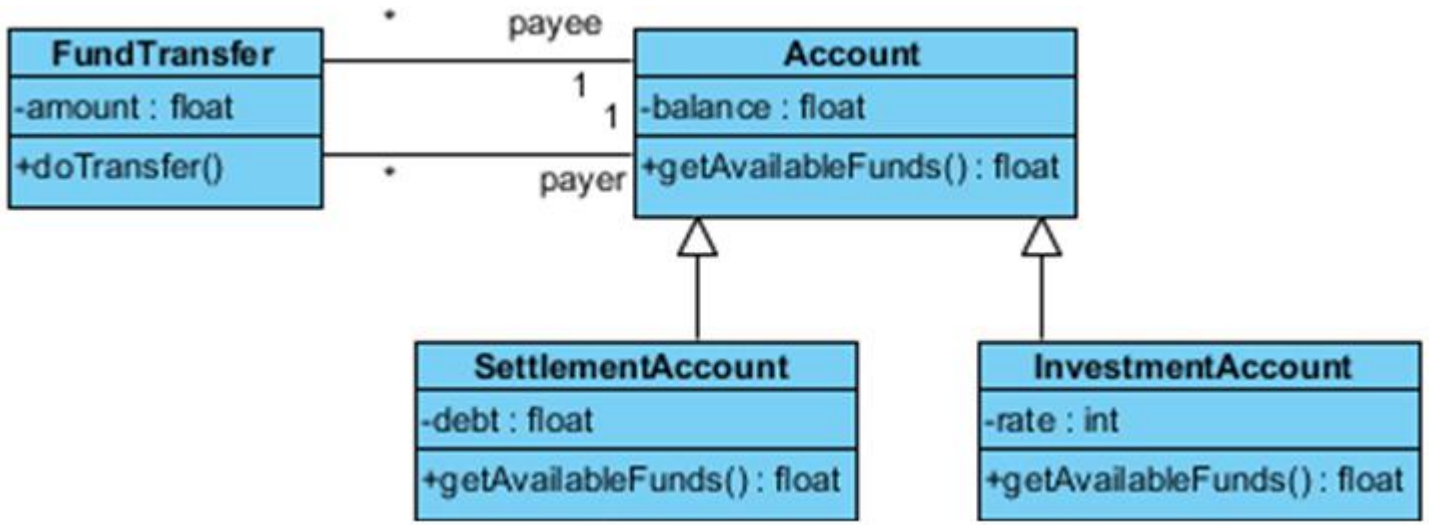
Q1

- 1. Look at Figure below. To cater to more type of accounts, the Account class is to be modified to check for the type of accounts and provide the appropriate balance in the getAvailableFunds() method. However, the initial Account class has been working fine and it is advised that it should not be modified.

- Propose a design principle to apply and show the new design in UML Class Diagram.


FundTransfer
-amount : float
+doTransfer()
*
payee
1
*
1
payer
Account
-balance : float
+getAvailableFunds() : float
public float getAvailableFunds() {
return balance ;
}
modify to
Account
-balance : float
-isSettlementAcc : Boolean
-isInvestmentAcc : Boolean
-debt : float
-rate : int
+getAvailableFunds() : float
public float getAvailableFunds() {
if (isSettlementAcc)
return balance - debt ;
else if (isInvestmentAcc)
return balance *(1 + rate/100) ;
return balance;
}

Open-Closed Principle (OCP)

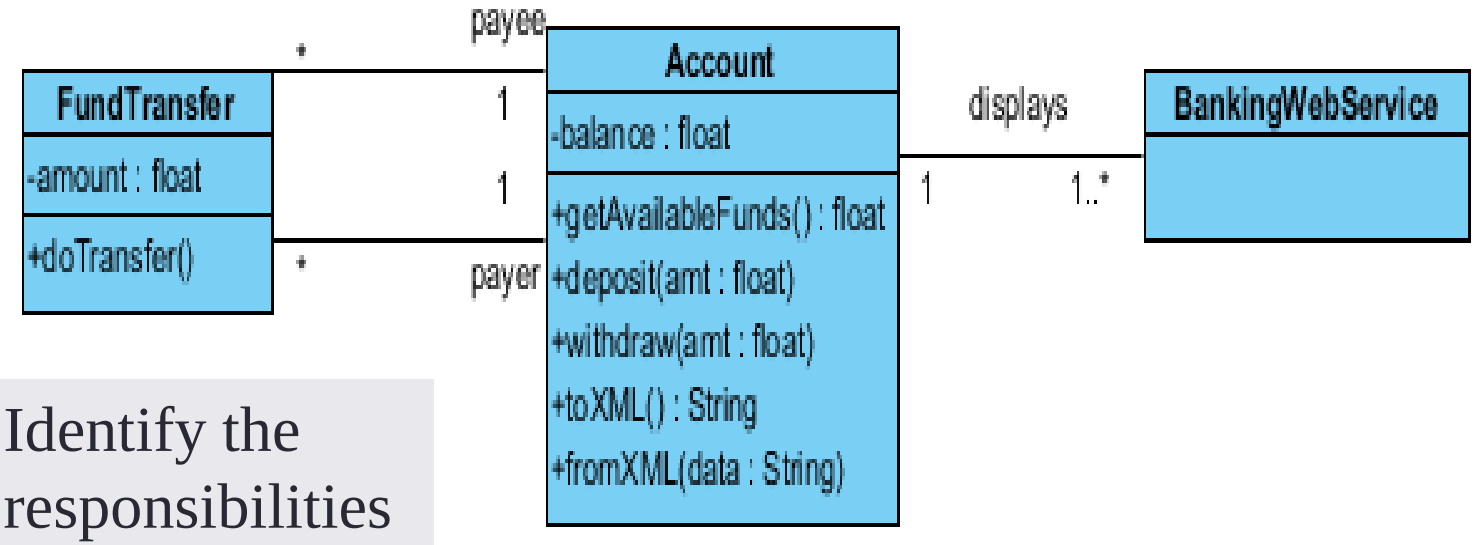



```
pubic float getAvailableFunds(){
 return super.getAvailableFunds() – debt;
}
```

```
pubic float getAvailableFunds(){
 return super.getAvailableFunds()*(1+rate/100)
}
```

Q2

- Look at Figure below. The BankingWebService uses Account class toXML() and fromXML() methods to display and update the account details respectively.



a. Identify the responsibilities of the Account class.

Responsibilities:

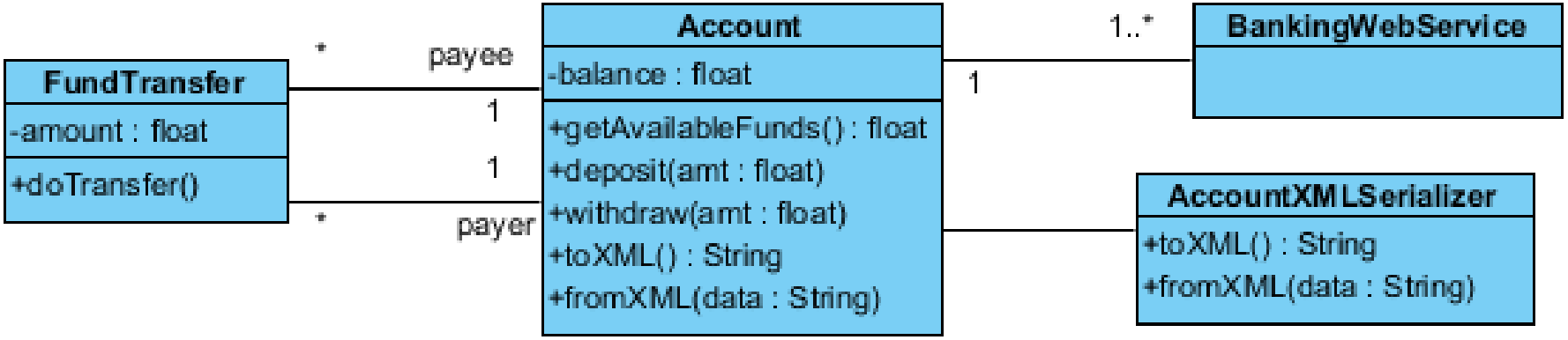
- model bank account transaction
- serialise account to/from XML string

⇨Two reasons why this class might need to

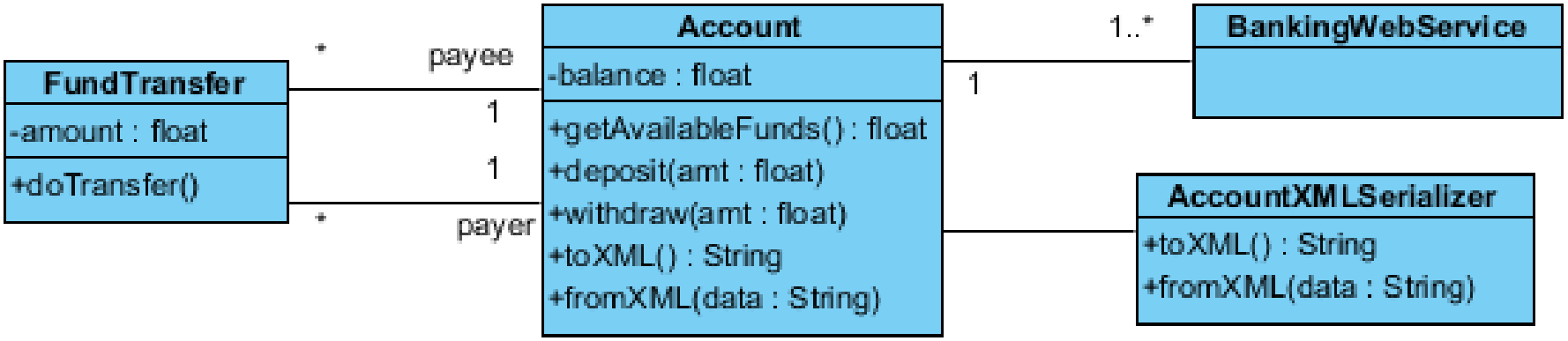
Single Responsibility Principle (SRP)

  - changes to domain logic
  - changes to XML format

b. Apply the Single Responsibility Principle to the class and show the new design in UML Class diagram.

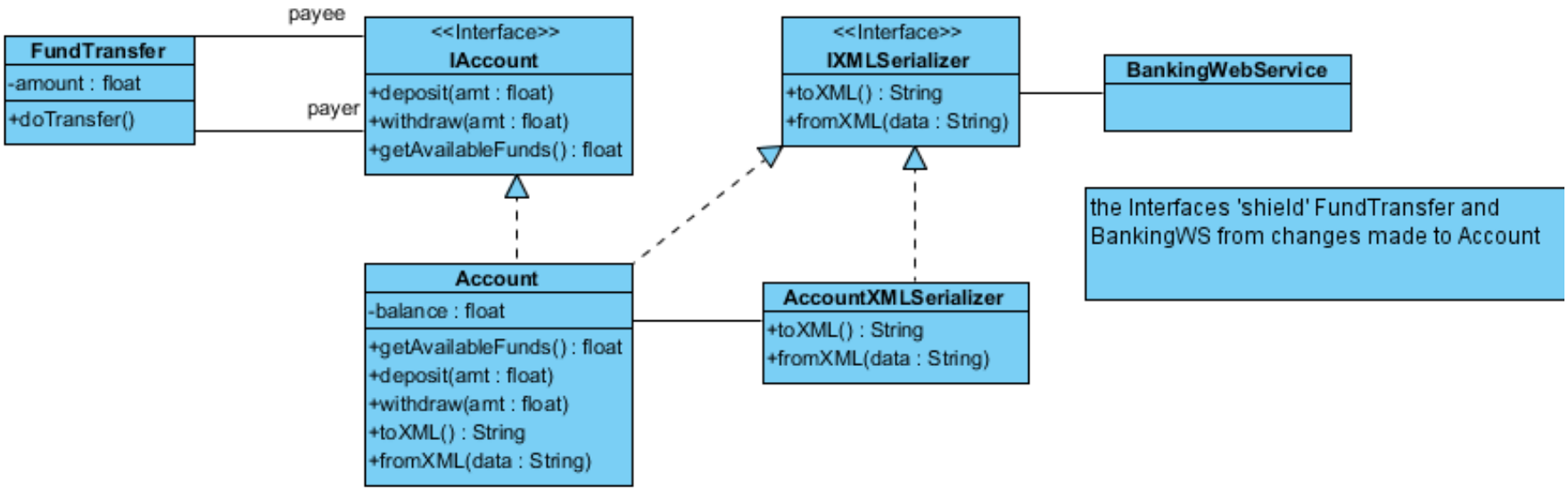


c. Explain what needs to be done if either of the deposit or withdraw methods is modified.



(c) Though BankingWebService does not use the 2 methods, but it is couple to the Account class. Any changes made to the Account class, BankingWebService class may need to be tested and recompiled.

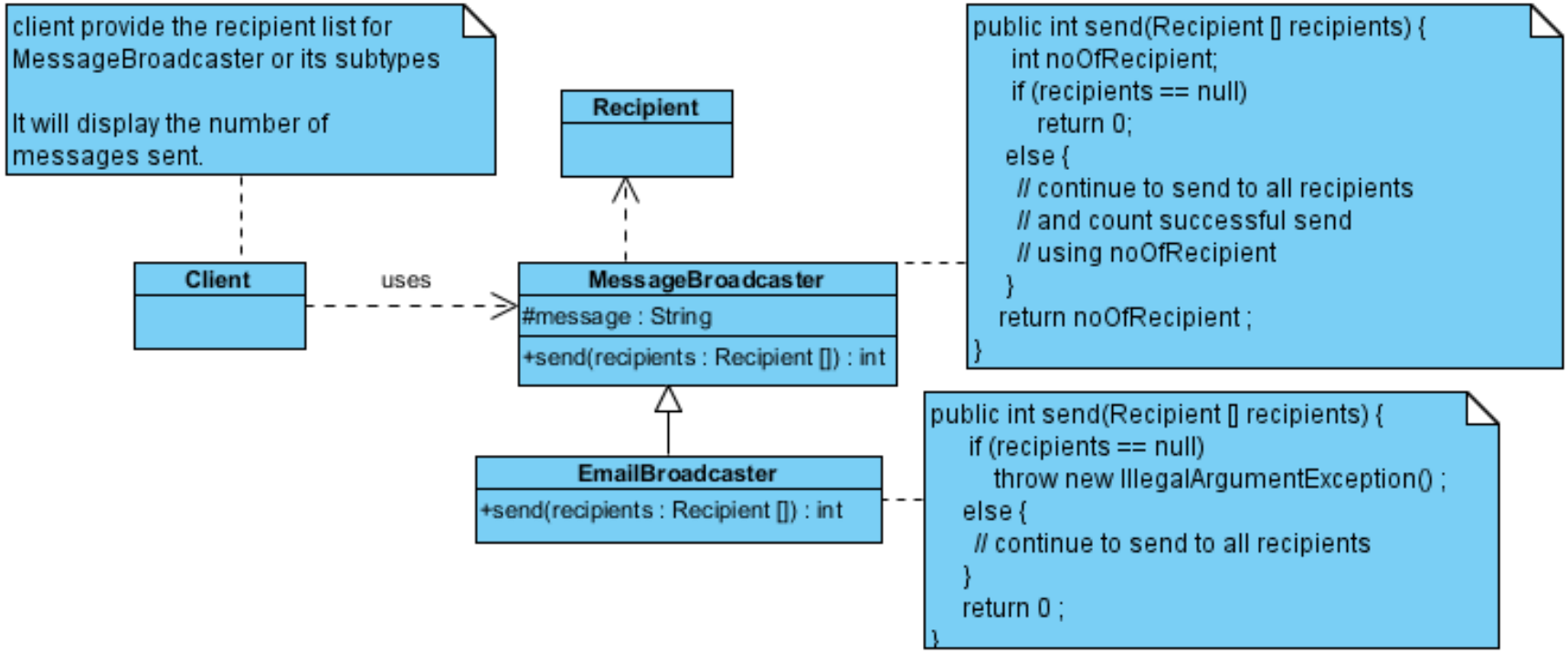
d. How can you improve (c)? Identify the principle used and show the solution in UML Class Diagram.



Dependency Injection Principle (DIP) and Interface Segregation Principle (ISP)

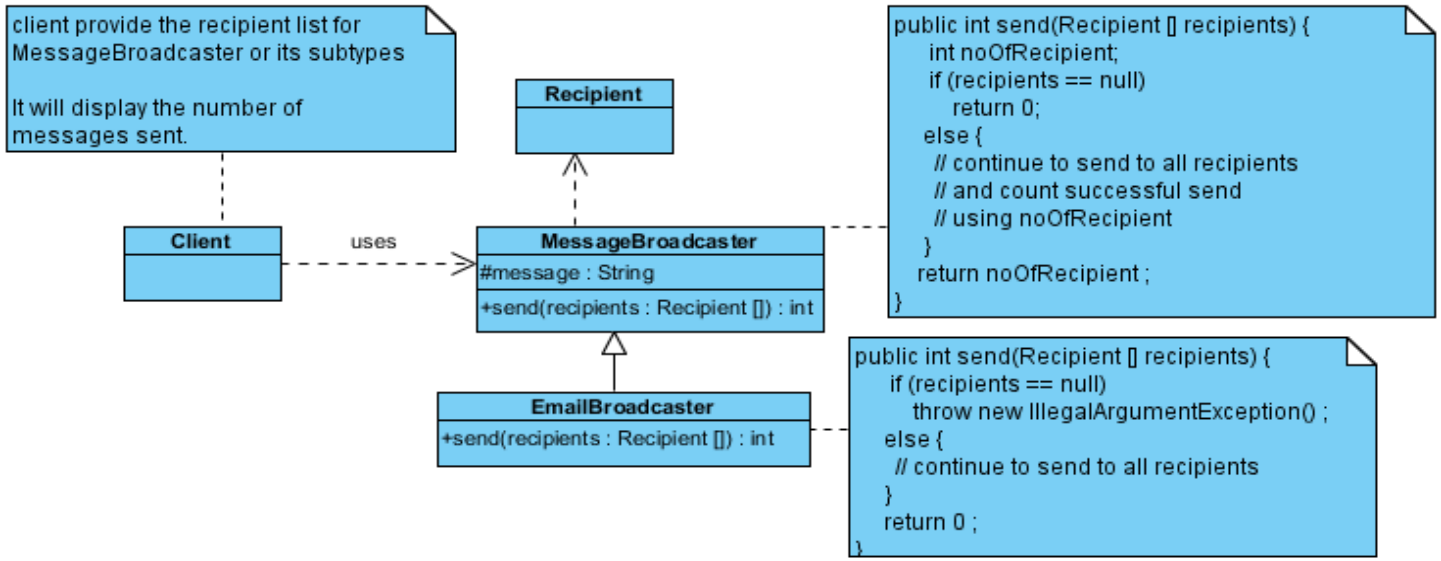
If different clients depend on different methods of the same class, then a change to one method might require a recompile and redeployment of other clients who use different methods. Creating several client-specific interfaces, one for each type of client, with the methods that type of client requires, reduces this problem significantly.

Q3. Look at Figure below. Using Liskov Substitution Principle (LSP), comment on the design.

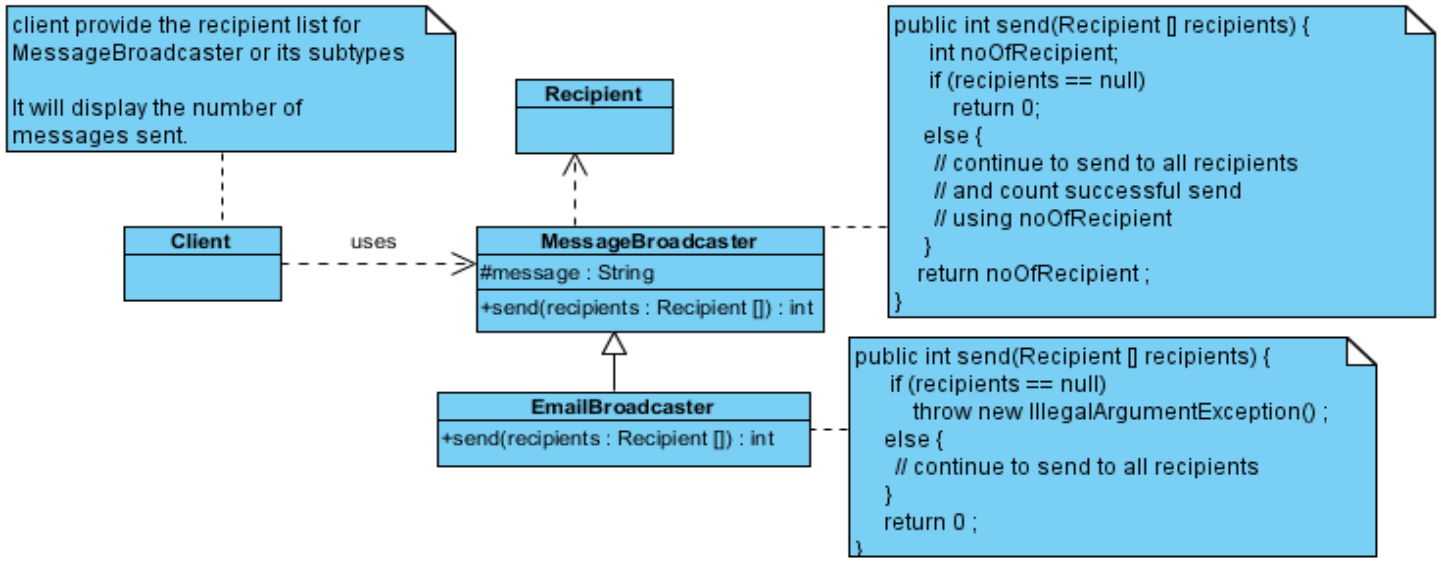


Subclass must do all the things super class do

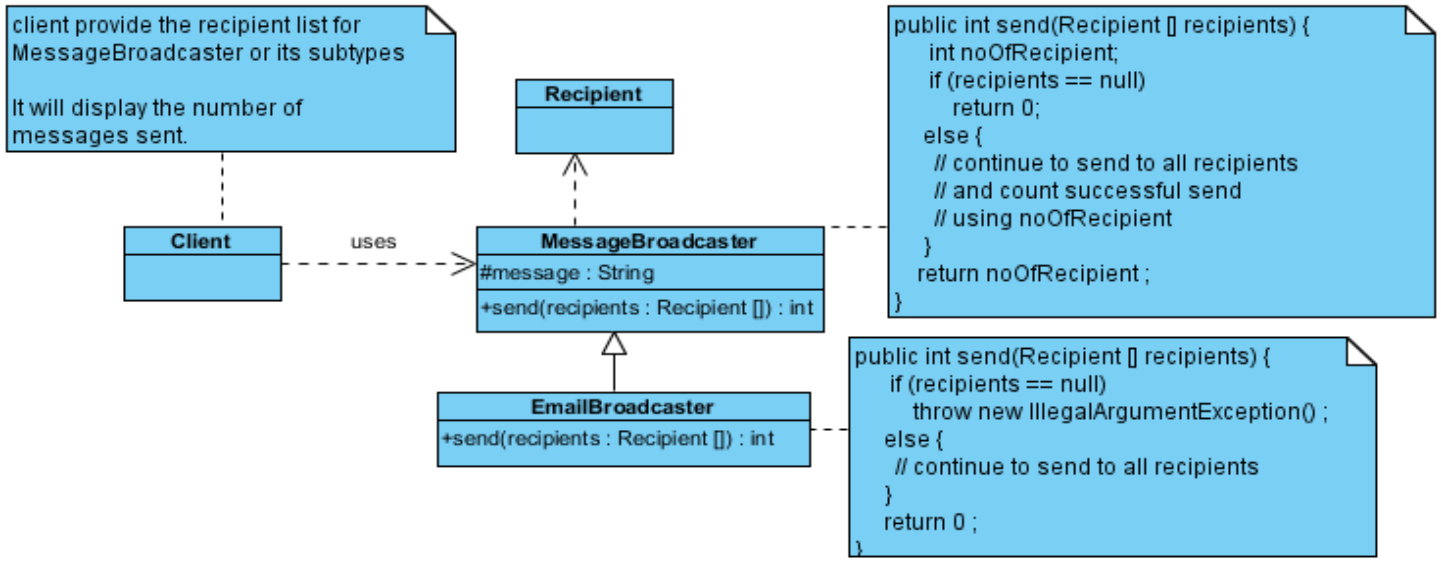
Subclass must not bring any trouble that super class don't



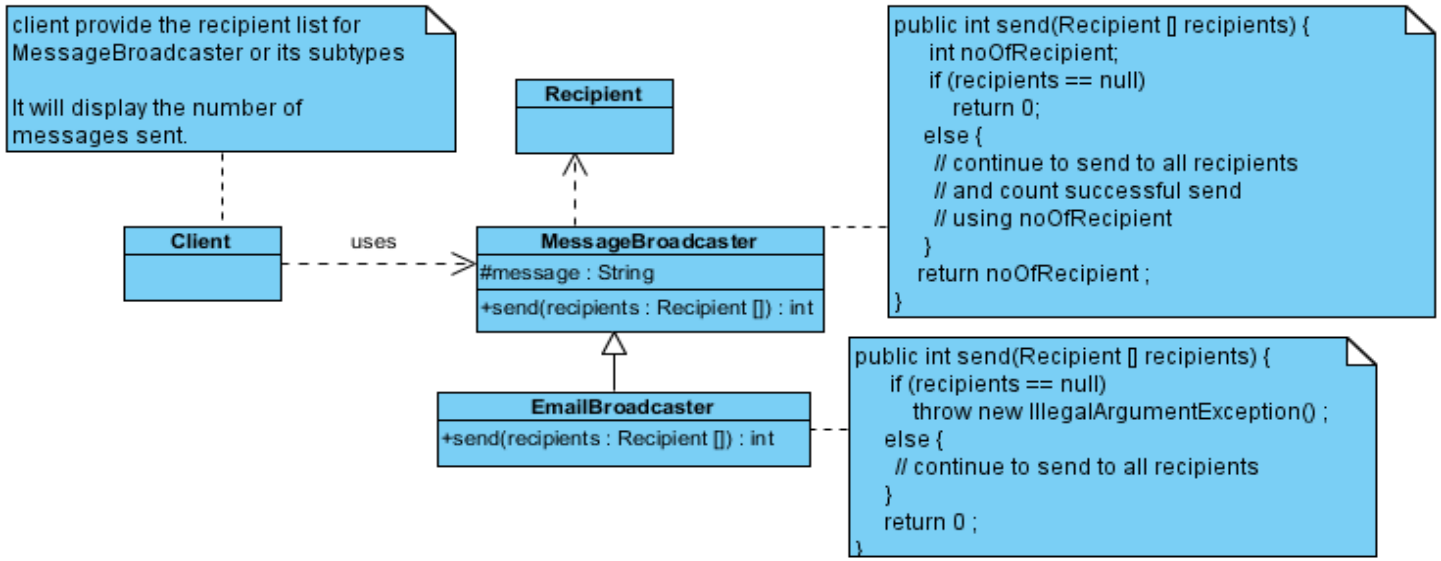
	superClass	subClass



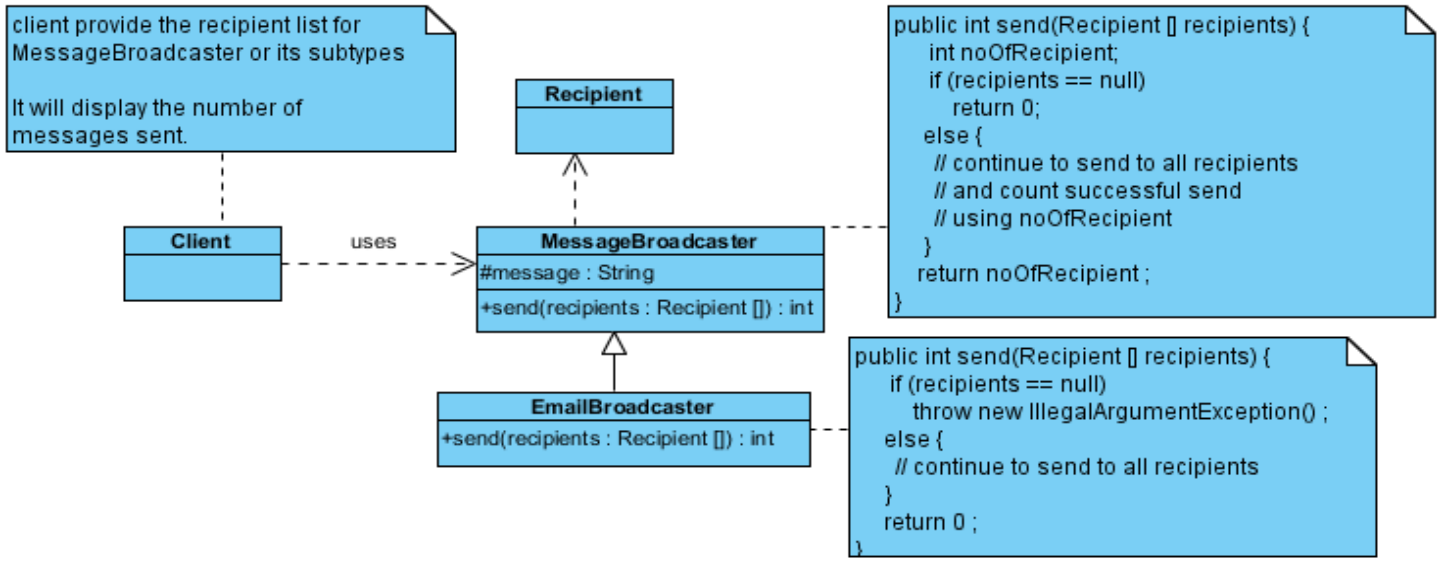
	superClass	subClass
If(recipients == null)		



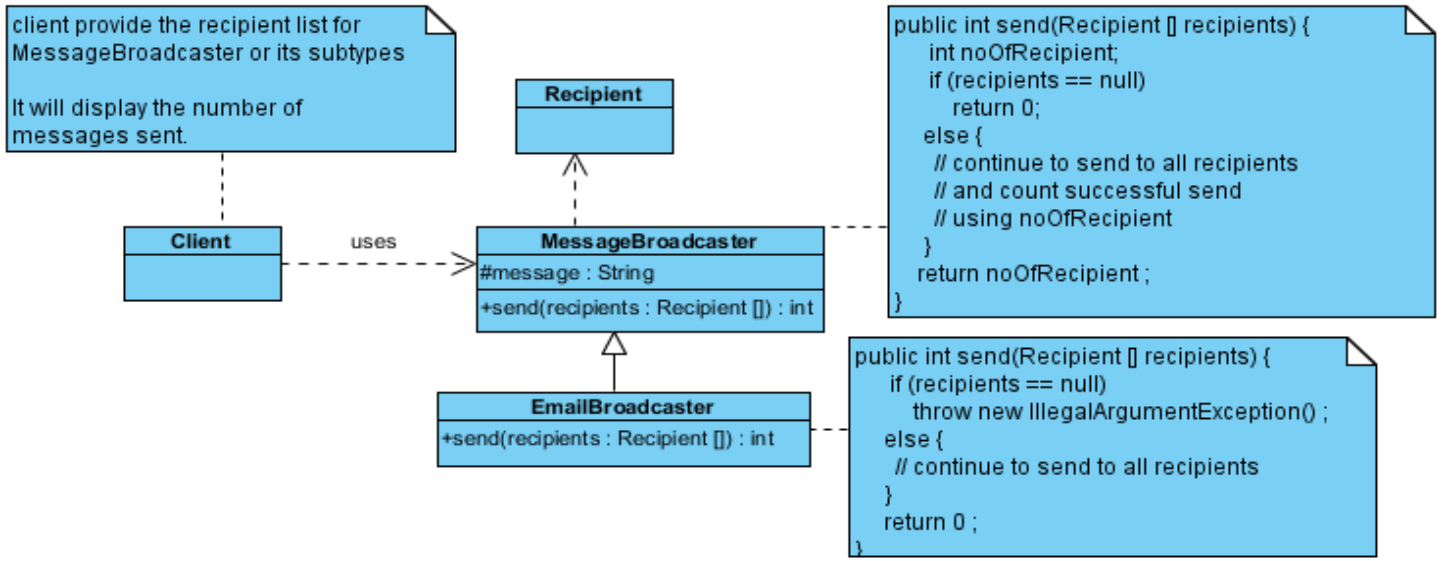
	superClass	subClass
If(recipients == null)	return 0	



	superClass	**subClass**
If(recipients == null)	return 0	Throw exception



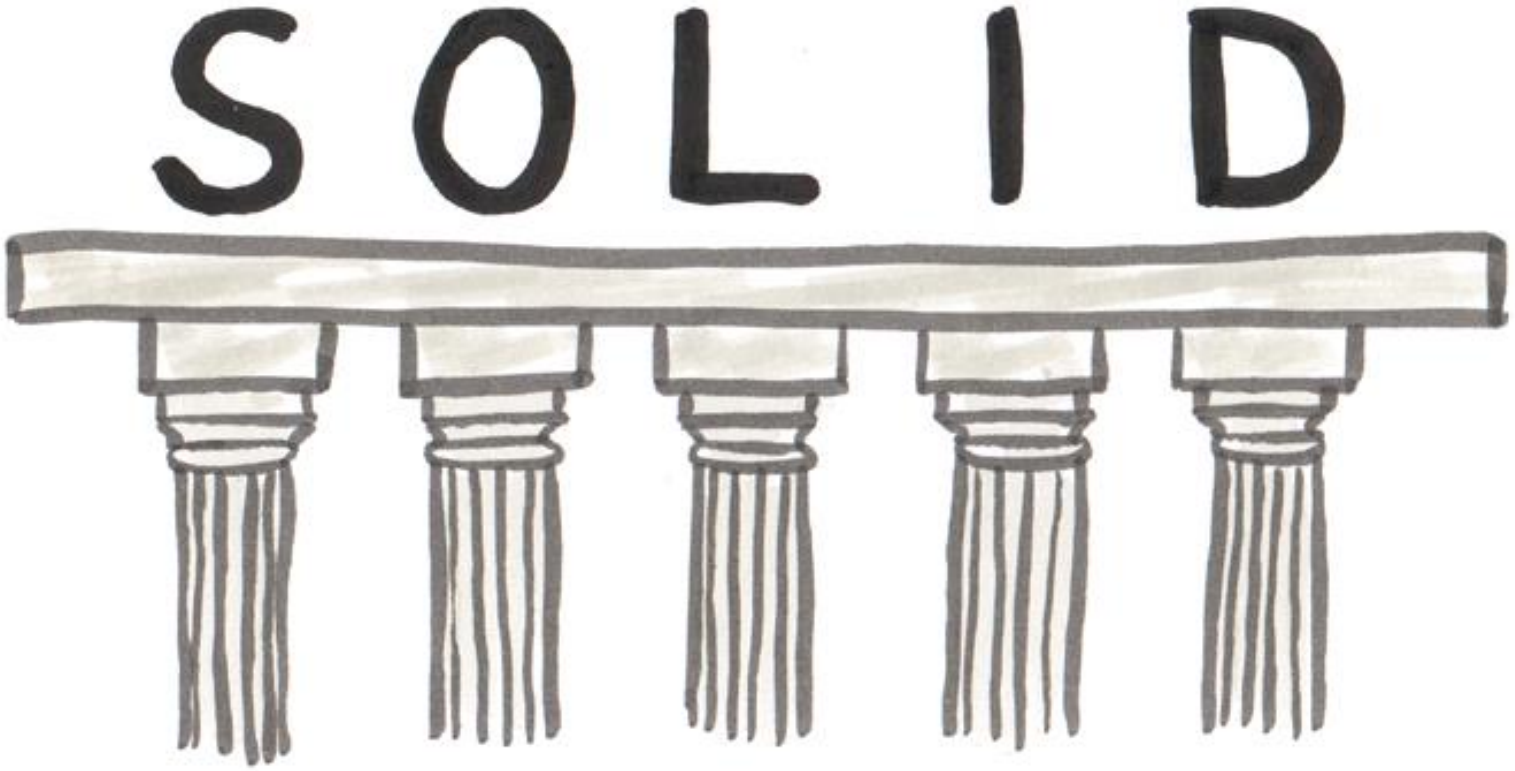
	superClass	**subClass**
If(recipients == null)	return 0	Throw exception
If(recipients != null)	Continue to send to all recipients	Continue to send to all recipients
	Count successful send	NA
	Using noOfRecipient	NA



	superClass	**subClass**
If(recipients == null)	return 0	Throw exception
If(recipients != null)	Continue to send to all recipients	Continue to send to all recipients
	Count successful send	NA
	Using noOfRecipient	NA
Return value if (recipients != null)	return noOfRecipient	return 0

- This principle states that Subclasses or derived classes should be completely substituted of superclass. The Subclass should enhance the functionality but not reduce it.
- Functions that use pointers and reference of base classes must be able to use objects derived classes without knowing it.

Liskov Substitution Principle (LSP)


SOLID