고급 객체지향 개발론

06.

Spring DI + Test tools

daumkakao
최윤상

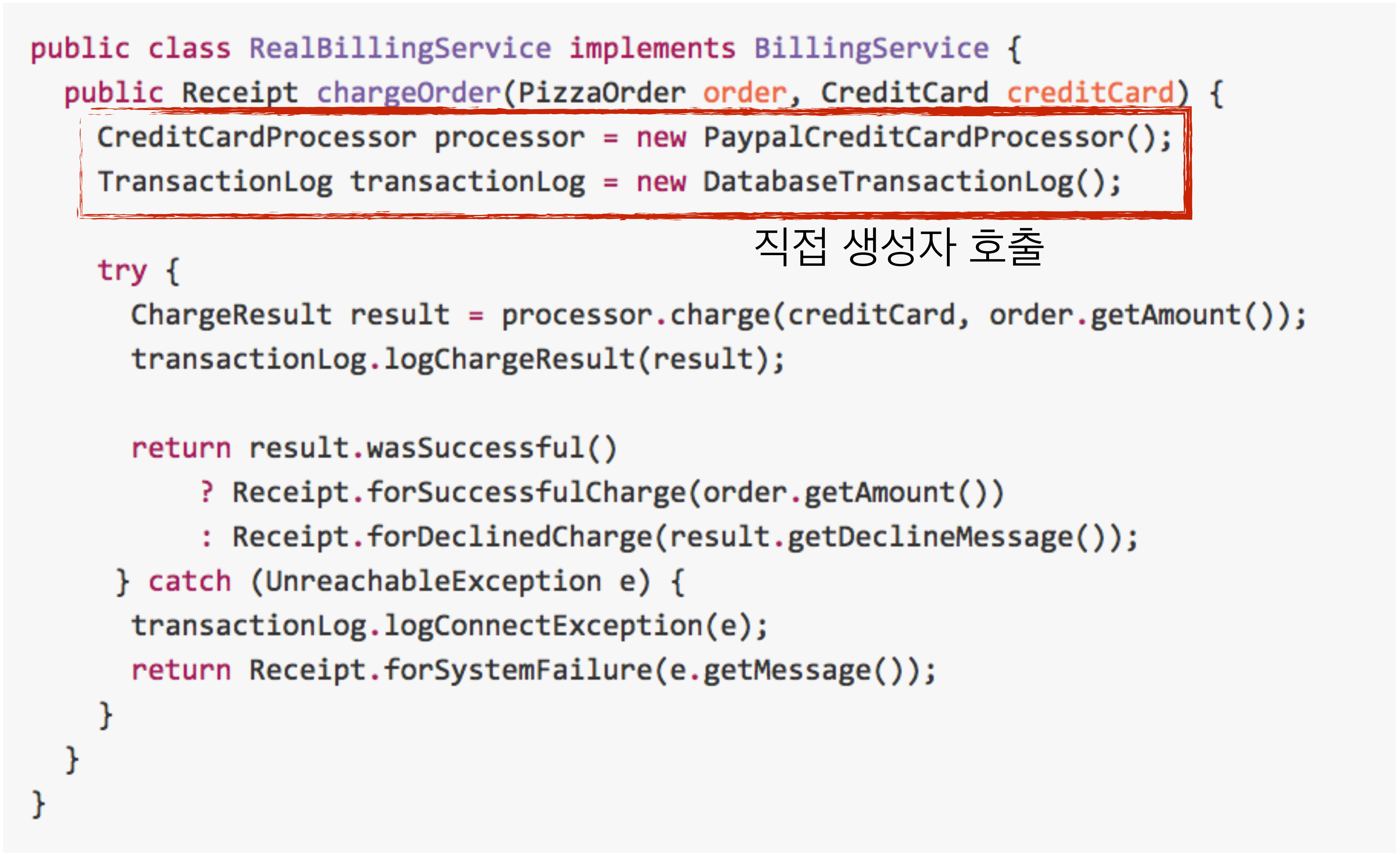
Dependency Controll

피자주문 서비스

```java
public interface BillingService {

  /**
   * Attempts to charge the order to the credit card. Both successful and
   * failed transactions will be recorded.
   *
   * @return a receipt of the transaction. If the charge was successful, the
   *      receipt will be successful. Otherwise, the receipt will contain a
   *      decline note describing why the charge failed.
   */
  Receipt chargeOrder(PizzaOrder order, CreditCard creditCard);
}
```

https://github.com/google/guice/wiki/Motivation

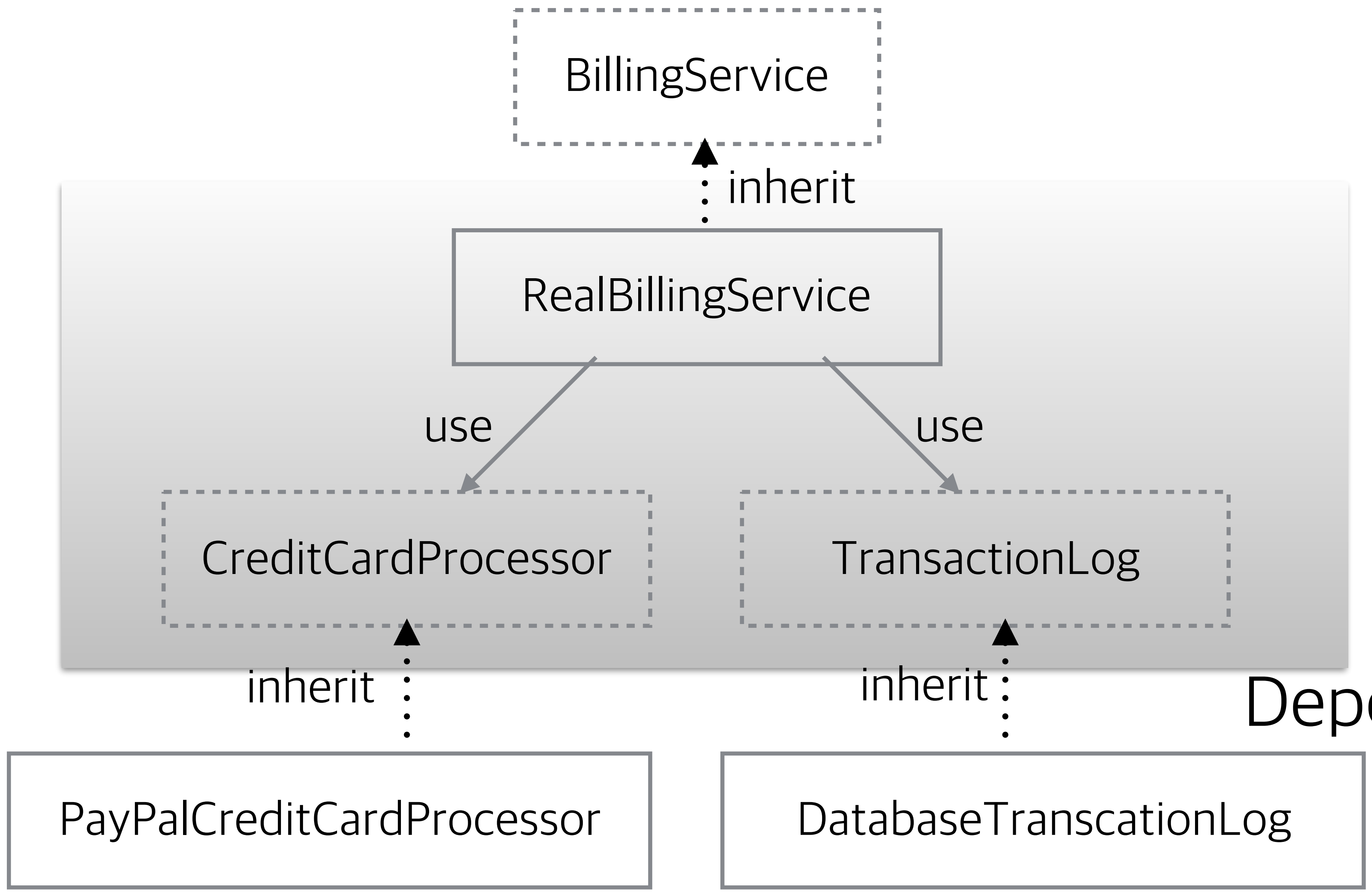
구현클래스



```java
public class RealBillingService implements BillingService {
  public Receipt chargeOrder(PizzaOrder order, CreditCard creditCard) {
    CreditCardProcessor processor = new PaypalCreditCardProcessor();
    TransactionLog transactionLog = new DatabaseTransactionLog();

    try {
      ChargeResult result = processor.charge(creditCard, order.getAmount());
      transactionLog.logChargeResult(result);

      return result.wasSuccessful()
          ? Receipt.forSuccessfulCharge(order.getAmount())
          : Receipt.forDeclinedCharge(result.getDeclineMessage());
     } catch (UnreachableException e) {
      transactionLog.logConnectException(e);
      return Receipt.forSystemFailure(e.getMessage());
    }
  }
}
```

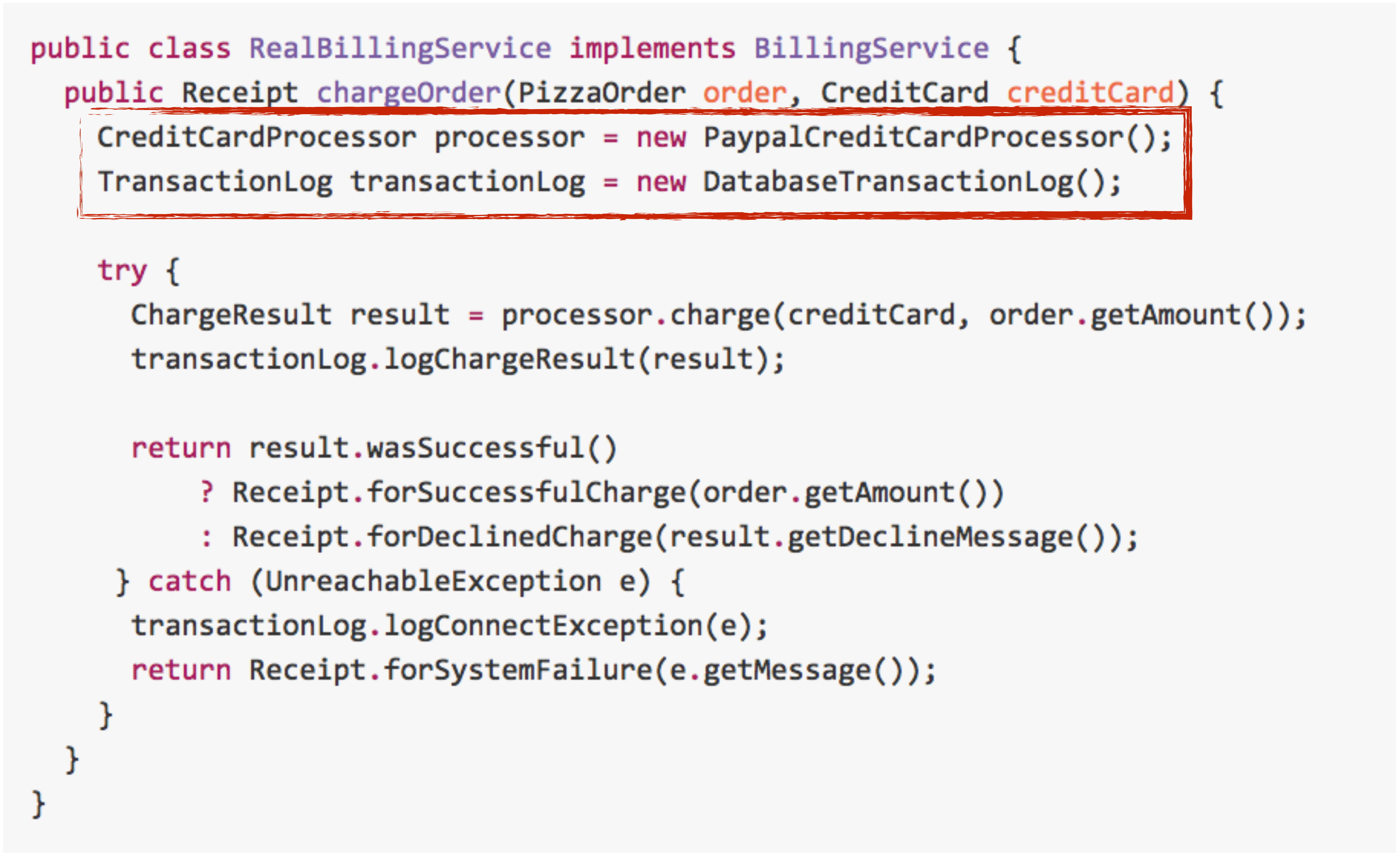
직접 생성자 호출

https://github.com/google/guice/wiki/Motivation



Dependency Inversion

problem#1.직접 생성자를 호출하므로써 저수준 구현에 의존하게 됨
problem#2. 객체에 대한 테스트가 어려움



```
public class RealBillingService implements BillingService {
  public Receipt chargeOrder(PizzaOrder order, CreditCard creditCard) {
    CreditCardProcessor processor = new PaypalCreditCardProcessor();
    TransactionLog transactionLog = new DatabaseTransactionLog();

    try {
      ChargeResult result = processor.charge(creditCard, order.getAmount());
      transactionLog.logChargeResult(result);

      return result.wasSuccessful()
          ? Receipt.forSuccessfulCharge(order.getAmount())
          : Receipt.forDeclinedCharge(result.getDeclineMessage());
     } catch (UnreachableException e) {
      transactionLog.logConnectException(e);
      return Receipt.forSystemFailure(e.getMessage());
    }
  }
}
```

https://github.com/google/guice/wiki/Motivation

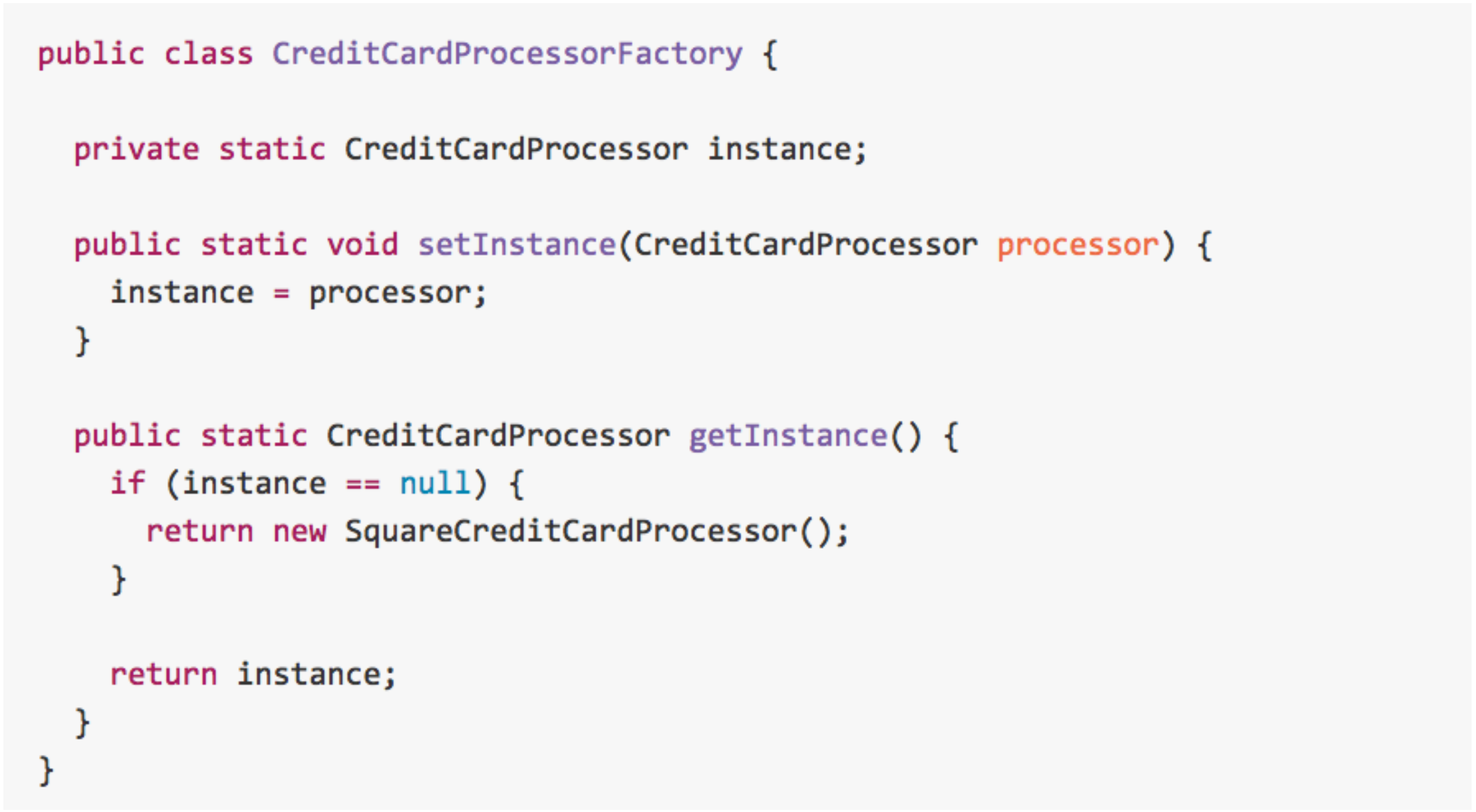
객체 생성에 따른

의존성을 어떻게 **제어**할까?

객체 생성의 책임(역할)을 분리하자!

Factories

factory클래스를 이용하여 구현클래스를 감추고 객체생성

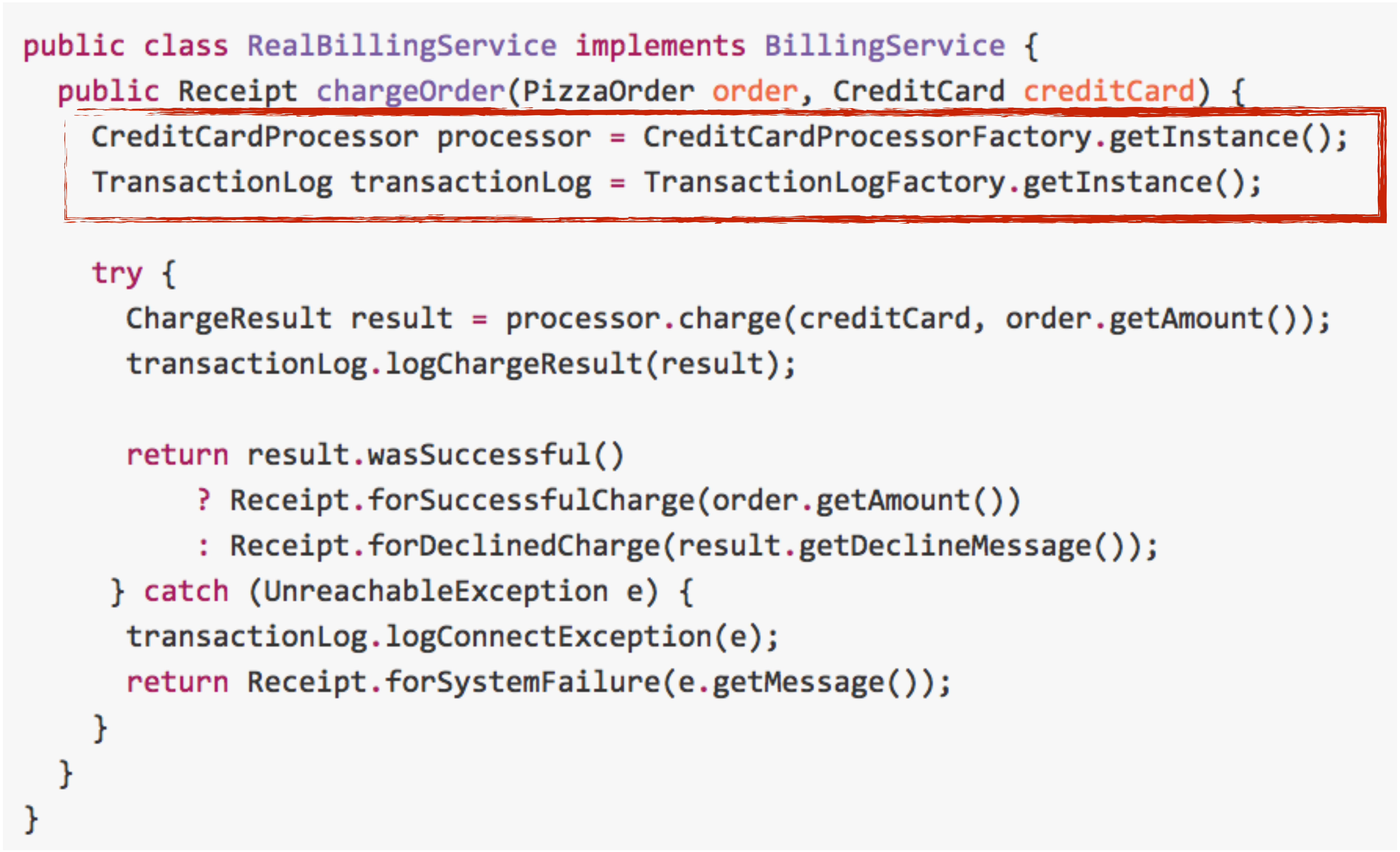


```
public class CreditCardProcessorFactory {

  private static CreditCardProcessor instance;

  public static void setInstance(CreditCardProcessor processor) {
    instance = processor;
  }

  public static CreditCardProcessor getInstance() {
    if (instance == null) {
      return new SquareCreditCardProcessor();
    }

    return instance;
  }
}
```

https://github.com/google/guice/wiki/Motivation

Factories

이제 BillingService는 구현 클래스를 모르게 됨

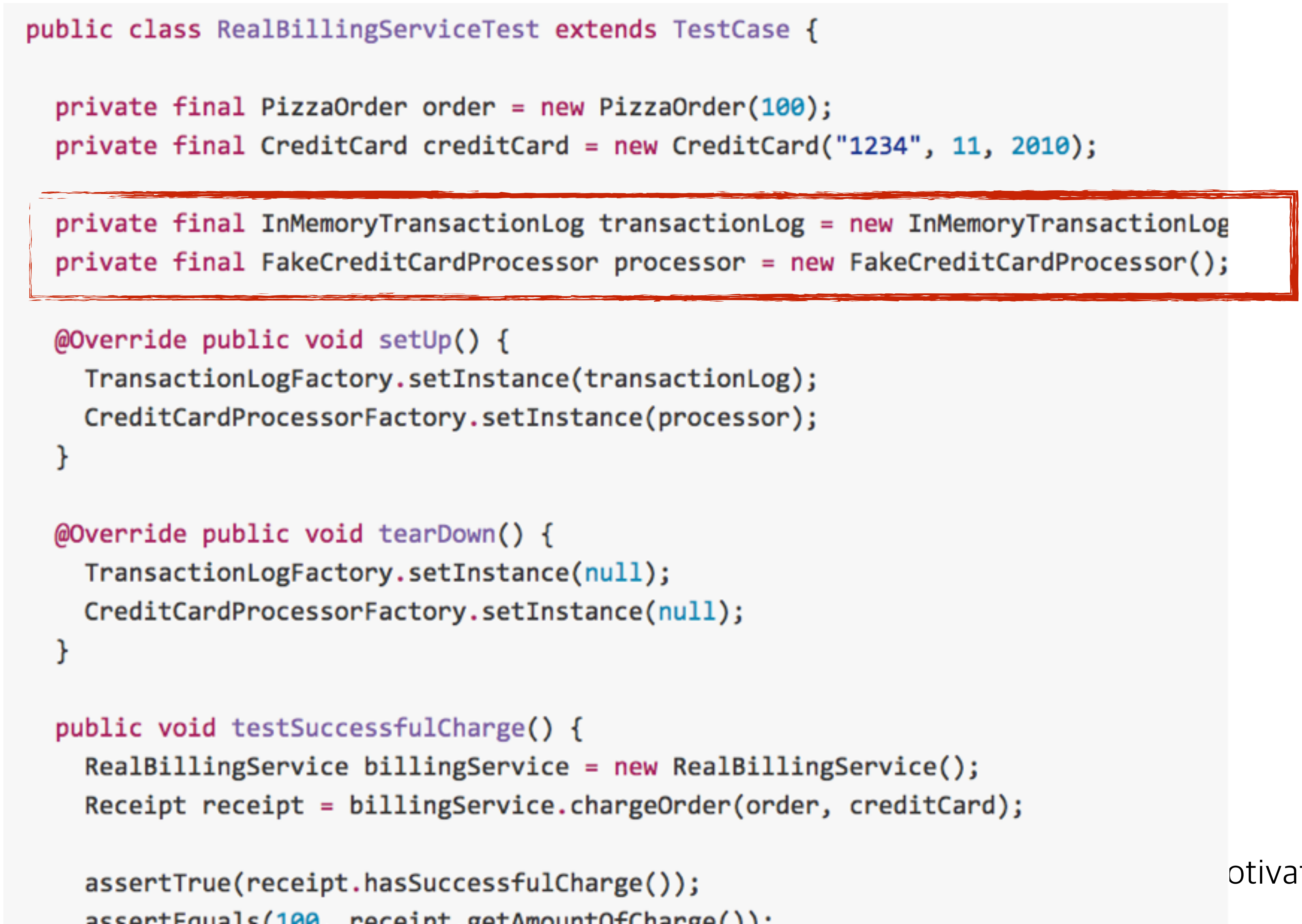


```
public class RealBillingService implements BillingService {
  public Receipt chargeOrder(PizzaOrder order, CreditCard creditCard) {
    CreditCardProcessor processor = CreditCardProcessorFactory.getInstance();
    TransactionLog transactionLog = TransactionLogFactory.getInstance();

    try {
      ChargeResult result = processor.charge(creditCard, order.getAmount());
      transactionLog.logChargeResult(result);

      return result.wasSuccessful()
          ? Receipt.forSuccessfulCharge(order.getAmount())
          : Receipt.forDeclinedCharge(result.getDeclineMessage());
     } catch (UnreachableException e) {
      transactionLog.logConnectException(e);
      return Receipt.forSystemFailure(e.getMessage());
    }
  }
}
```

https://github.com/google/guice/wiki/Motivation

Factories

mock을 설정해서 단위 테스트도 할 수 있음



```
public class RealBillingServiceTest extends TestCase {

  private final PizzaOrder order = new PizzaOrder(100);
  private final CreditCard creditCard = new CreditCard("1234", 11, 2010);

  private final InMemoryTransactionLog transactionLog = new InMemoryTransactionLog
  private final FakeCreditCardProcessor processor = new FakeCreditCardProcessor();

  @Override public void setUp() {
    TransactionLogFactory.setInstance(transactionLog);
    CreditCardProcessorFactory.setInstance(processor);
  }

  @Override public void tearDown() {
    TransactionLogFactory.setInstance(null);
    CreditCardProcessorFactory.setInstance(null);
  }

  public void testSuccessfulCharge() {
    RealBillingService billingService = new RealBillingService();
    Receipt receipt = billingService.chargeOrder(order, creditCard);

    assertTrue(receipt.hasSuccessfulCharge());
    assertEquals(100, receipt.getAmountOfCharge());
```

Factory를 이용한 의존성제어의 문제

전역변수로 mock구현체를 제공하므로 주의가 필요
e.g. tearDown 실패

가장 큰 문제는 의존성 이슈가 코드 안쪽에 숨겨져 있음

https://github.com/google/guice/wiki/Motivation

Dependency Injection

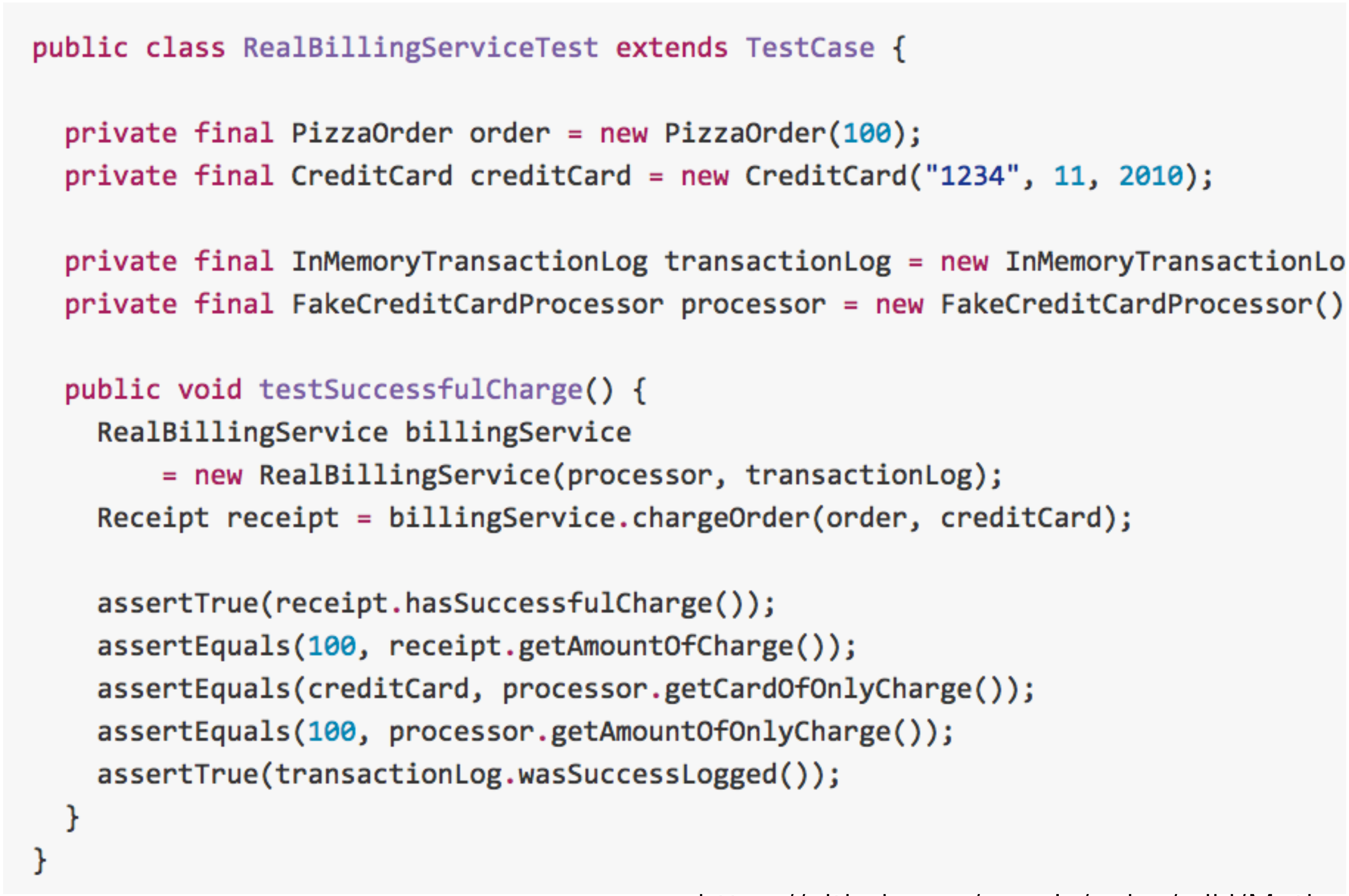
생성자로 주입

```
public class RealBillingService implements BillingService {
  private final CreditCardProcessor processor;
  private final TransactionLog transactionLog;

  public RealBillingService(CreditCardProcessor processor,
      TransactionLog transactionLog) {
    this.processor = processor;
    this.transactionLog = transactionLog;
  }

  public Receipt chargeOrder(PizzaOrder order, CreditCard creditCard) {
    try {
      ChargeResult result = processor.charge(creditCard, order.getAmount());
      transactionLog.logChargeResult(result);

      return result.wasSuccessful()
          ? Receipt.forSuccessfulCharge(order.getAmount())
          : Receipt.forDeclinedCharge(result.getDeclineMessage());
     } catch (UnreachableException e) {
      transactionLog.logConnectException(e);
      return Receipt.forSystemFailure(e.getMessage());
    }
```

https://github.com/google/guice/wiki/Motivation

UnitTest

tearDown이 필요없음



```java
public class RealBillingServiceTest extends TestCase {

  private final PizzaOrder order = new PizzaOrder(100);
  private final CreditCard creditCard = new CreditCard("1234", 11, 2010);

  private final InMemoryTransactionLog transactionLog = new InMemoryTransactionLo
  private final FakeCreditCardProcessor processor = new FakeCreditCardProcessor()

  public void testSuccessfulCharge() {
    RealBillingService billingService
        = new RealBillingService(processor, transactionLog);
    Receipt receipt = billingService.chargeOrder(order, creditCard);

    assertTrue(receipt.hasSuccessfulCharge());
    assertEquals(100, receipt.getAmountOfCharge());
    assertEquals(creditCard, processor.getCardOfOnlyCharge());
    assertEquals(100, processor.getAmountOfOnlyCharge());
    assertTrue(transactionLog.wasSuccessLogged());
  }
}
```

https://github.com/google/guice/wiki/Motivation

의존성 주입의 유형

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

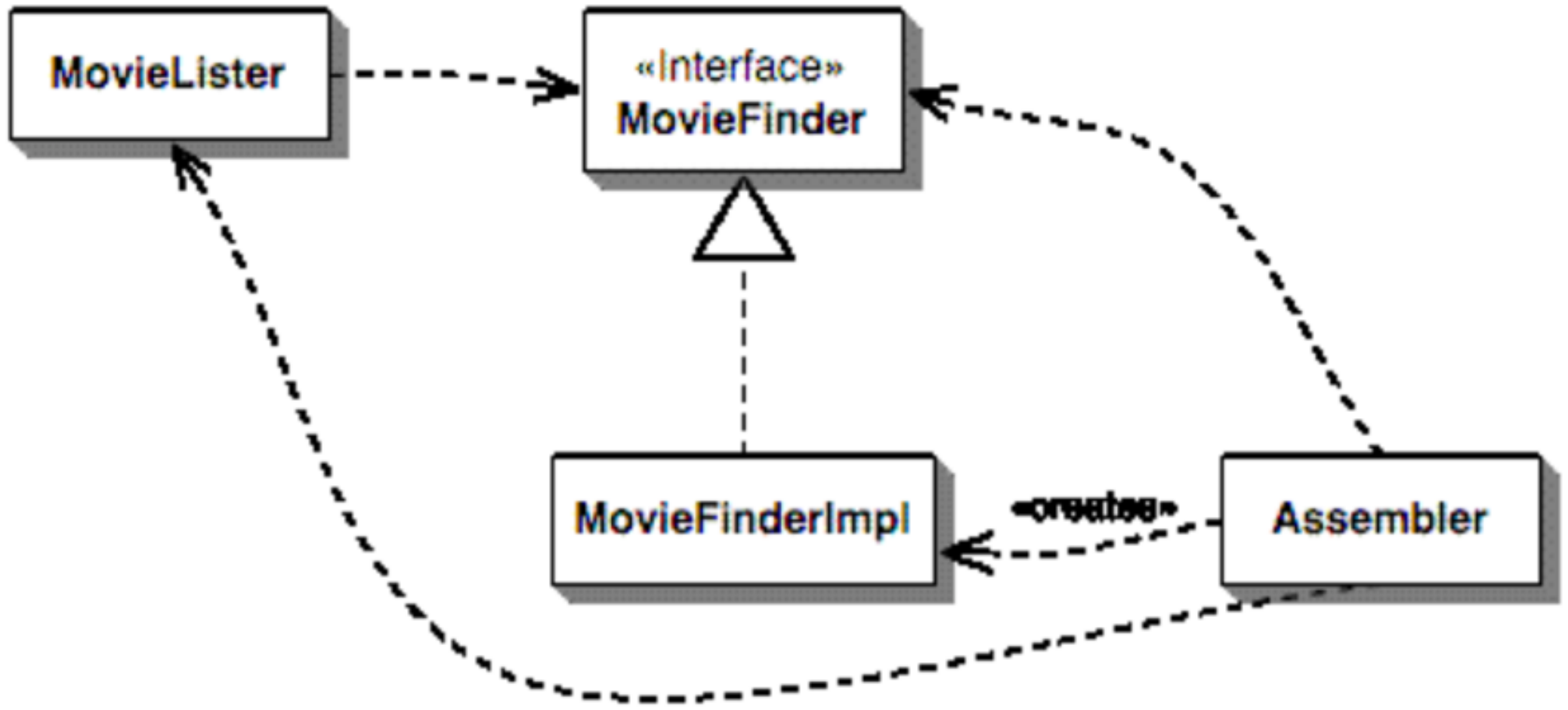
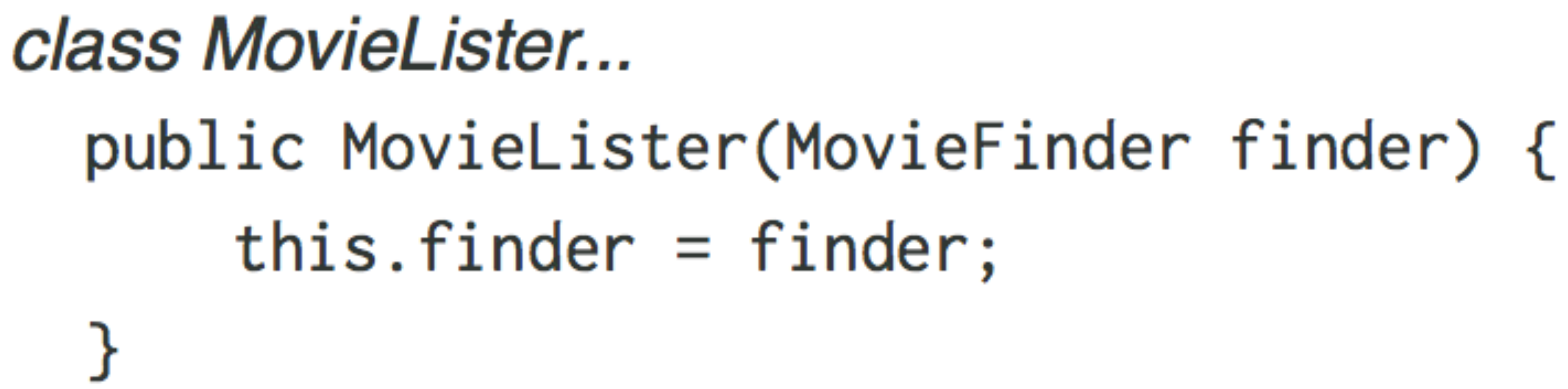


Figure 2: The dependencies for a Dependency Injector

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

#1
Constructor Injection

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection



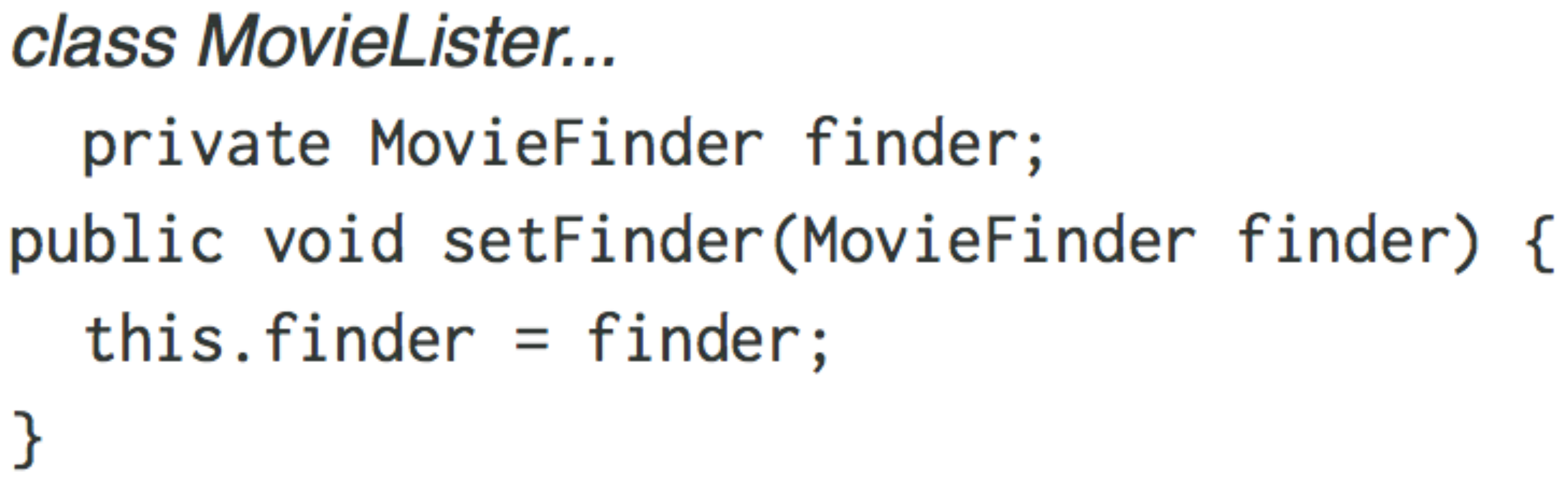
class MovieLister...

```
  public MovieLister(MovieFinder finder) {
      this.finder = finder;
  }
```

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

#2
Setter Injection

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection



class MovieLister...

```
  private MovieFinder finder;
public void setFinder(MovieFinder finder) {
  this.finder = finder;
}
```

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

#3
Interface Injection

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

```
public interface InjectFinder {
    void injectFinder(MovieFinder finder);
}
```

class MovieLister implements InjectFinder

```
 public void injectFinder(MovieFinder finder) {
     this.finder = finder;
 }
```

http://www.martinfowler.com/articles/injection.html#FormsOfDependencyInjection

Spring DI

Spring DI 용어들

Spring Bean / POJO

Inversion of Controll

IoC Container

Spring DI 실습

Unit Test 관련 Library

가독성 높은 테스트 작성을 위한 도구

Hamcrest

Matchers that can be combined to create flexible expressions of intent

https://github.com/hamcrest/JavaHamcrest

```
assertThat(obj, matcher)
```

술어*(predicate)*

https://github.com/hamcrest/JavaHamcrest

JUnit	Hamcrest
assertEquals(o1, o2);	*assertThat(o1, is(o2));*
assertNull(o1);	*assertThat(o1, nullValue());*
assertNotNull(o1);	*assertThat(o1, notNullValue());*
	assertThat(o1, not(nullValue()));
assertTrue(o instanceof Class);	*assertThat(o instanceOf(Class));*
assertEquals(c.size(), 3);	*assertThat(c, hasSize(3));*

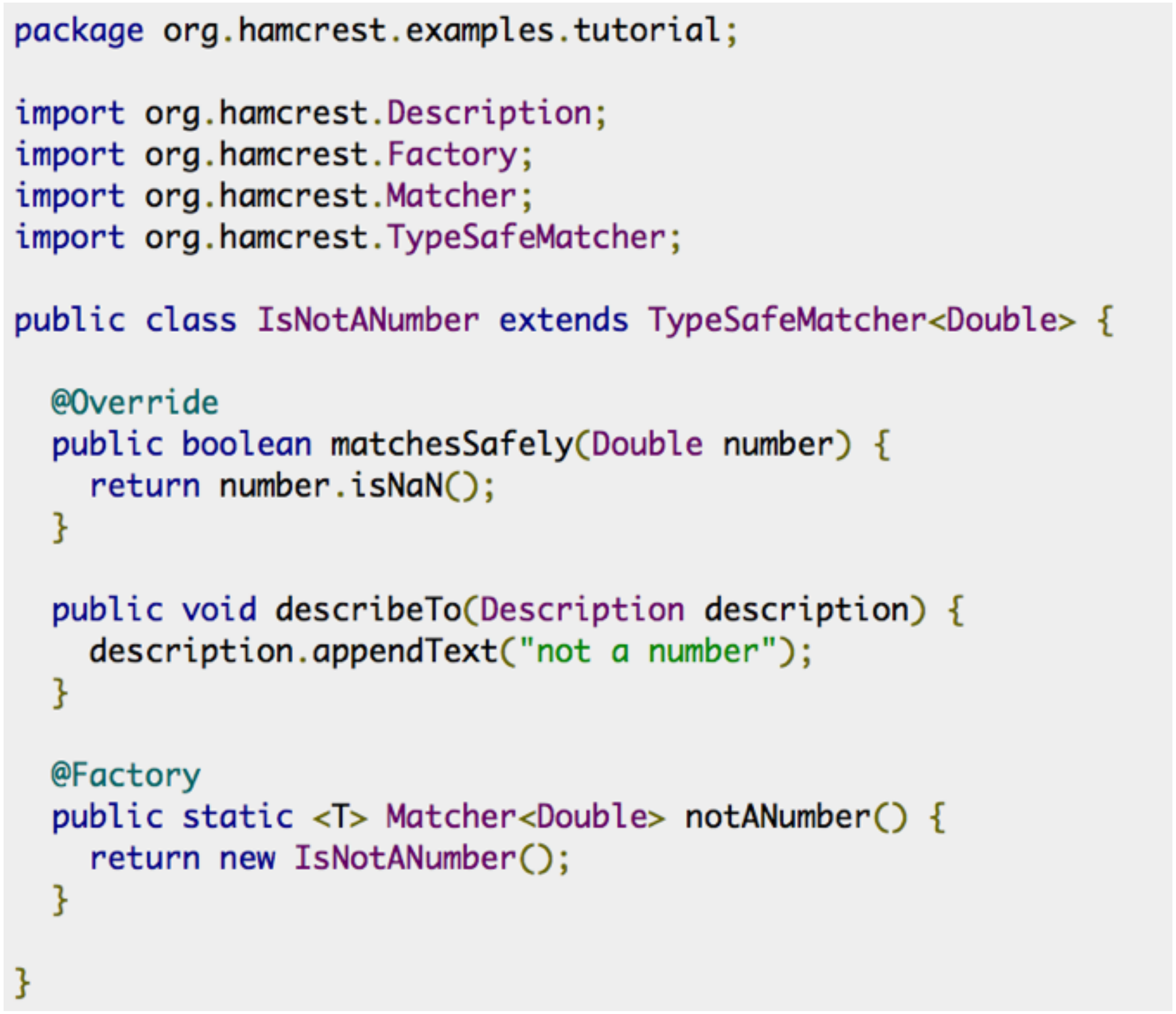
다양한 기본 *Matcher*제공

```
assertThat(2, greaterThan(1));
assertThat(1, lessThanOrEquals(2));
assertThat(“Believe”, containsString(“lie”);
assertThat(“hello”, equalToIgnoringCase(“HELLO”));
assertThat(o1,samePropertyValuesAs(o2));
assertThat(o1, hasProperty(“name”));
```

Custom Matcher를 만들수 있음

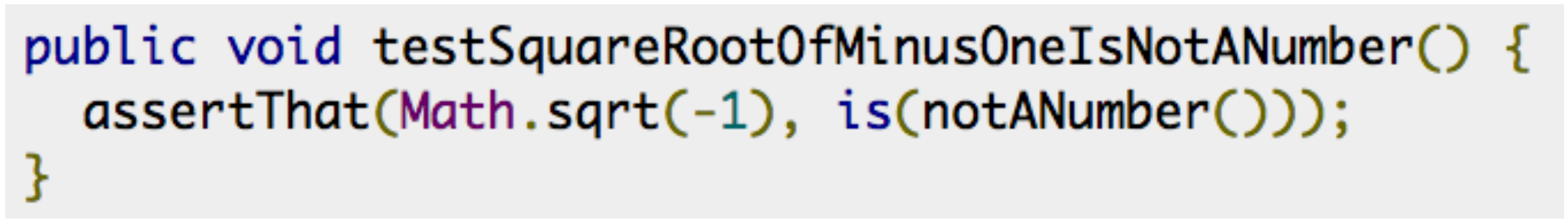
```
assertThat(person, is(student()));

assertThat(email, not(fromHell));

assertThat(person, hasNoFriends());
```

Example: Custom Matcher

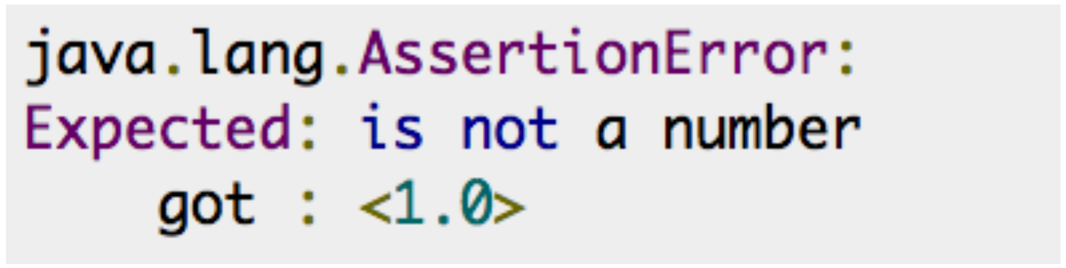


```java
package org.hamcrest.examples.tutorial;

import org.hamcrest.Description;
import org.hamcrest.Factory;
import org.hamcrest.Matcher;
import org.hamcrest.TypeSafeMatcher;

public class IsNotANumber extends TypeSafeMatcher<Double> {

  @Override
  public boolean matchesSafely(Double number) {
    return number.isNaN();
  }

  public void describeTo(Description description) {
    description.appendText("not a number");
  }

  @Factory
  public static <T> Matcher<Double> notANumber() {
    return new IsNotANumber();
  }

}
```

Example: Custom Matcher



```java
public void testSquareRootOfMinusOneIsNotANumber() {
  assertThat(Math.sqrt(-1), is(notANumber()));
}
```



```
java.lang.AssertionError:
Expected: is not a number
     got : <1.0>
```

Hamcrest

Domain의 용어를 사용하여

보다 **가독성있는 테스트코드**를

작성할수 있다

```
assertThat(맷데이먼, is(민폐캐릭()));
```

Java Unit Test를 위한 mocking Framework

Mocks aren't Stubs - Martin Fowler

번역 http://testing.jabberstory.net

Classical TDD vs. Mockist TDD

상태기반테스트 vs. 행위기반테스트

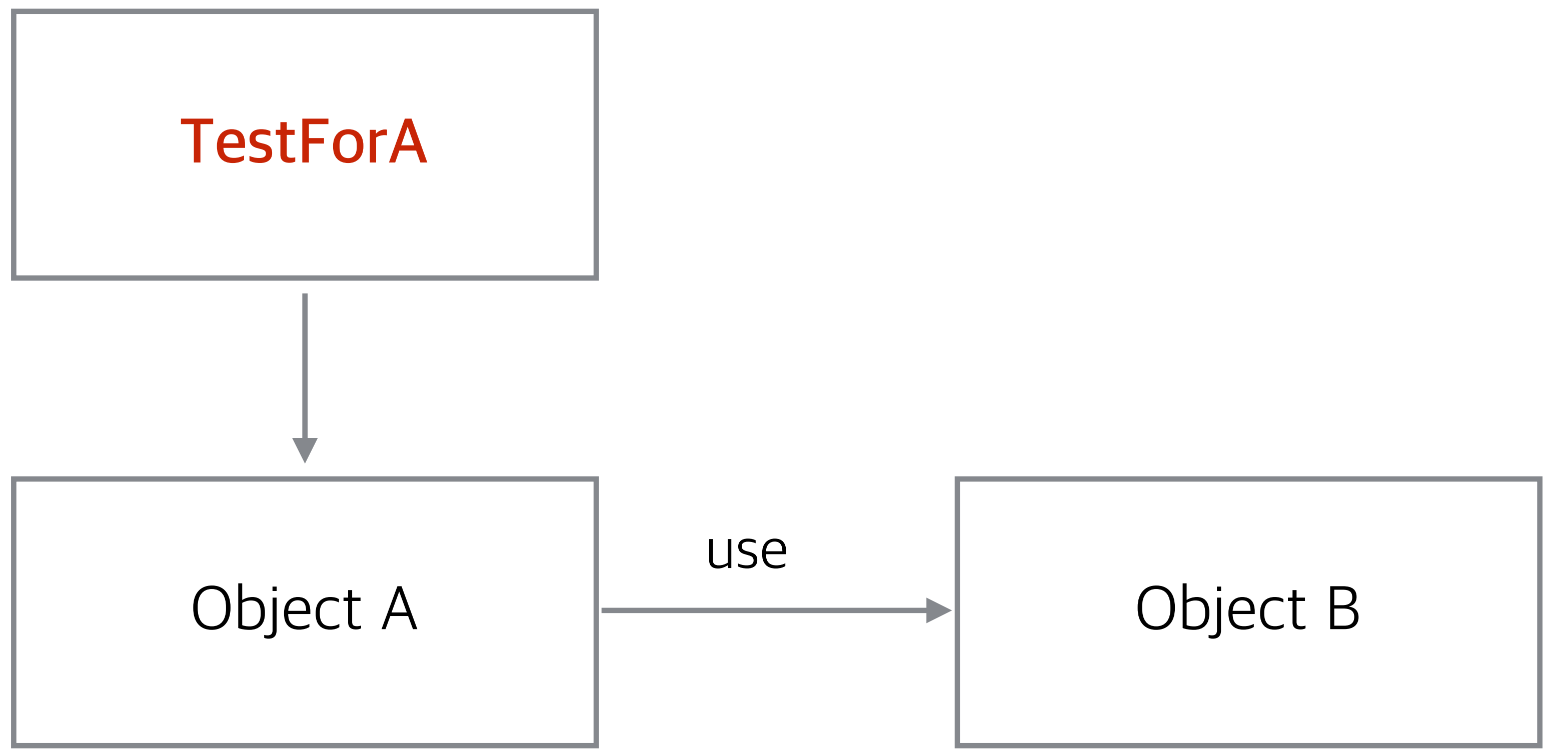
Test double의 종류 - Meszaros

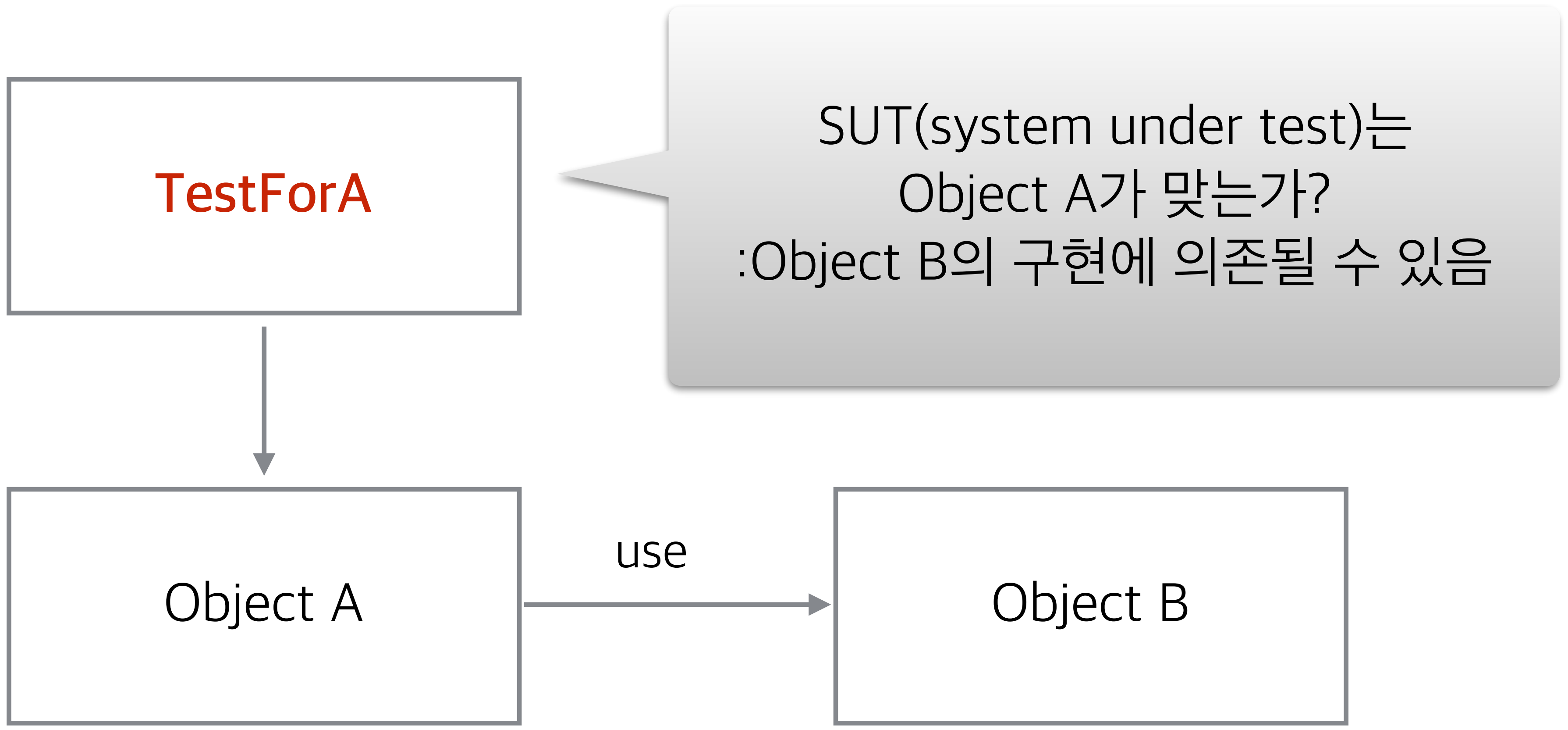
dummy: 전달하지만 실제 사용되지 않는 객체

fake: 동작하는 구현은 있지만, 실사용은 불가능
e.g. in-memory db

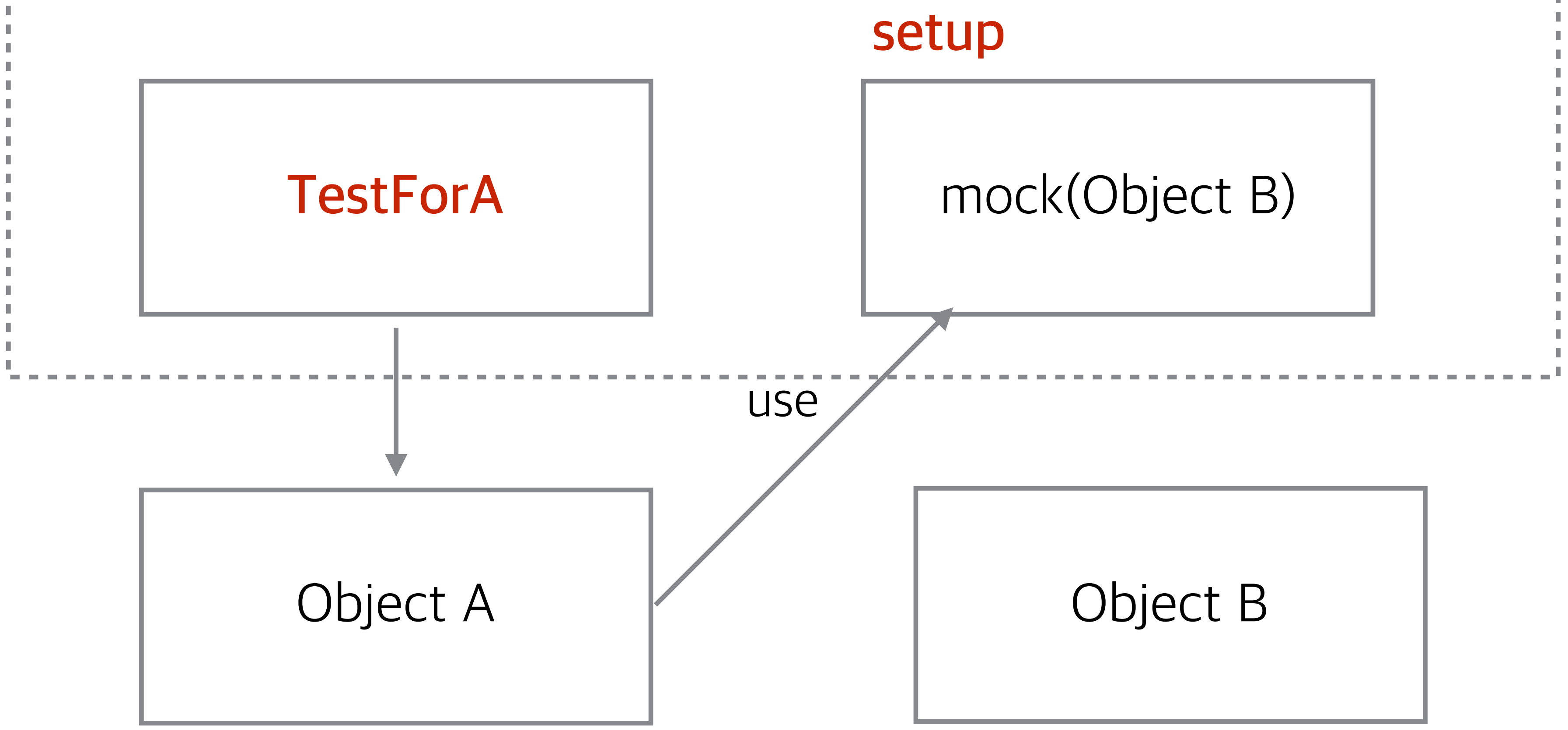
stubs: 미리 준비된 응답을 하는 형태.
테스트시 필요한 것만 구현됨

mocks: 수신하기를 기대하는 호출의 명세(specification)인
예측으로 미리 프로그램 된 객체


TestForA
Object A
use
Object B


TestForA
SUT(system under test)는
Object A가 맞는가?
:Object B의 구현에 의존될 수 있음
Object A
use
Object B

mocking test

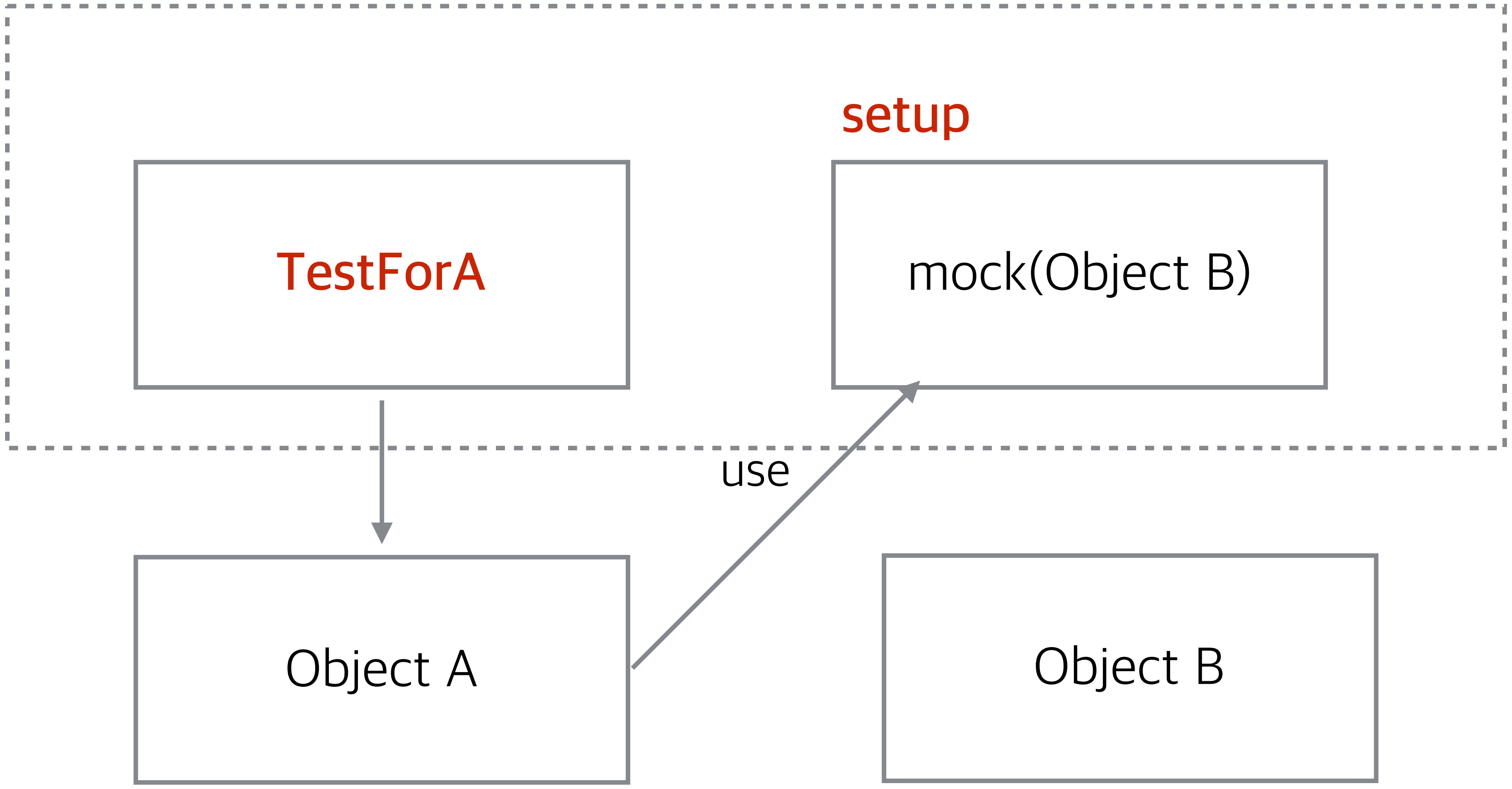


Pros.

objA의 기능을 테스트하는데 집중할 수 있음

objB의 변경에 의해 test가 깨지 않게 할 수 있음 (test isolation)

status가 없는 객체더라도 objB와의 interaction을 테스트할 수 있음

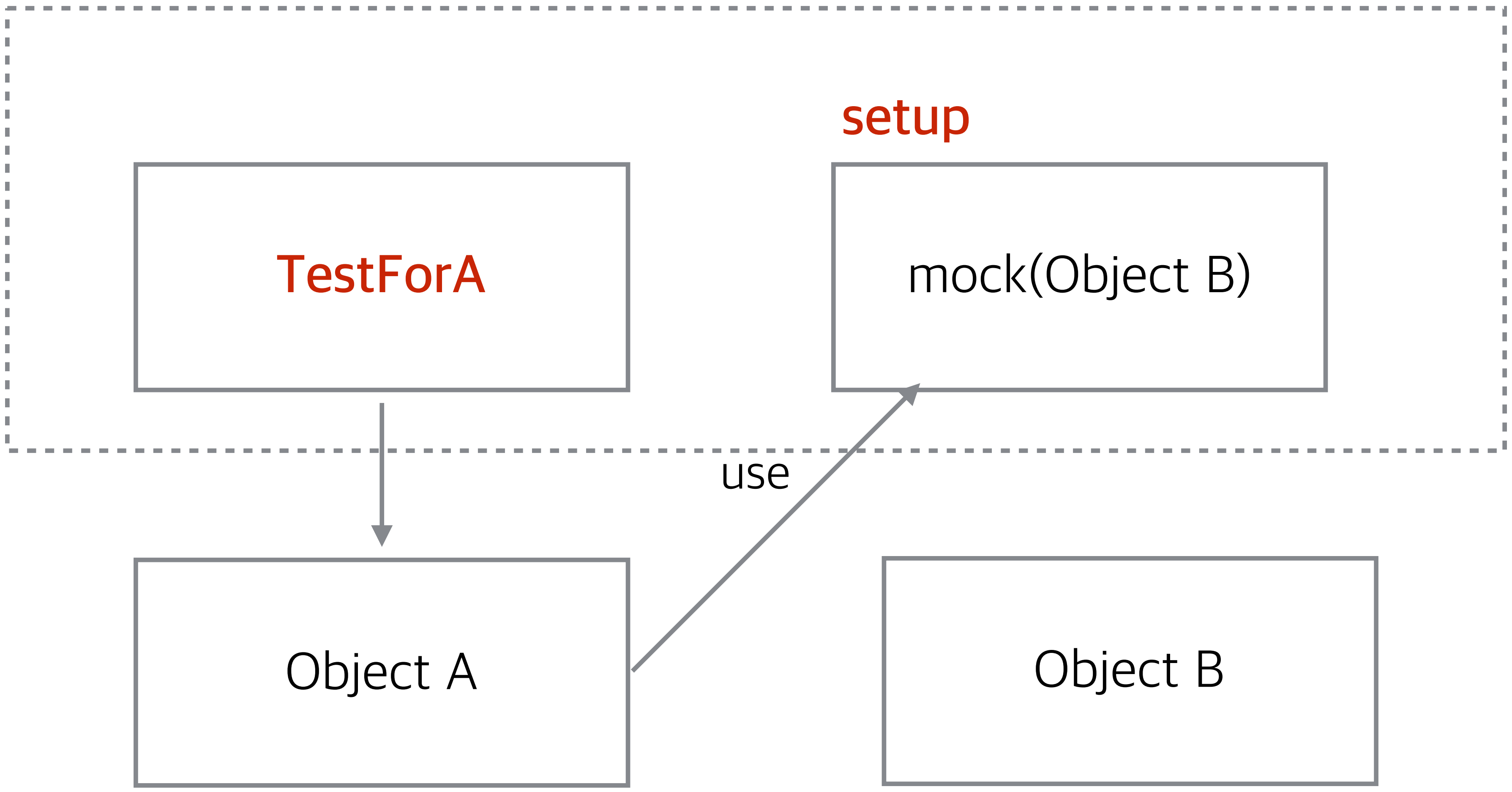


Cons.

interface를 분리해내고 그 interface로 mock구현체를 만들어야 함

mock 객체에서 spy를 구현의 부담이 있음

test 작성시 setup 단계가 길고 복잡해짐 (e.g. fixture)



mockito

빡치는 mocking을 편하게 해주는 도구

http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html