Design Defects and Restructuring

Engr. Abdul-Rahman Mahmood

abdulrahman@nu.edu.pk

alphapeeler.sf.net/pubkeys/pkey.htm

pk.linkedin.com/in/armahmood

www.twitter.com/alphapeeler

www.facebook.com/alphapeeler

abdulmahmood-sss alphasecure

armahmood786

http://alphapeeler.sf.net/me

alphapeeler#9321

reddit.com/user/alphapeeler

www.flickr.com/alphapeeler

http://alphapeeler.tumblr.com

armahmood786@jabber.org

alphapeeler@aim.com

mahmood_cubix 48660186

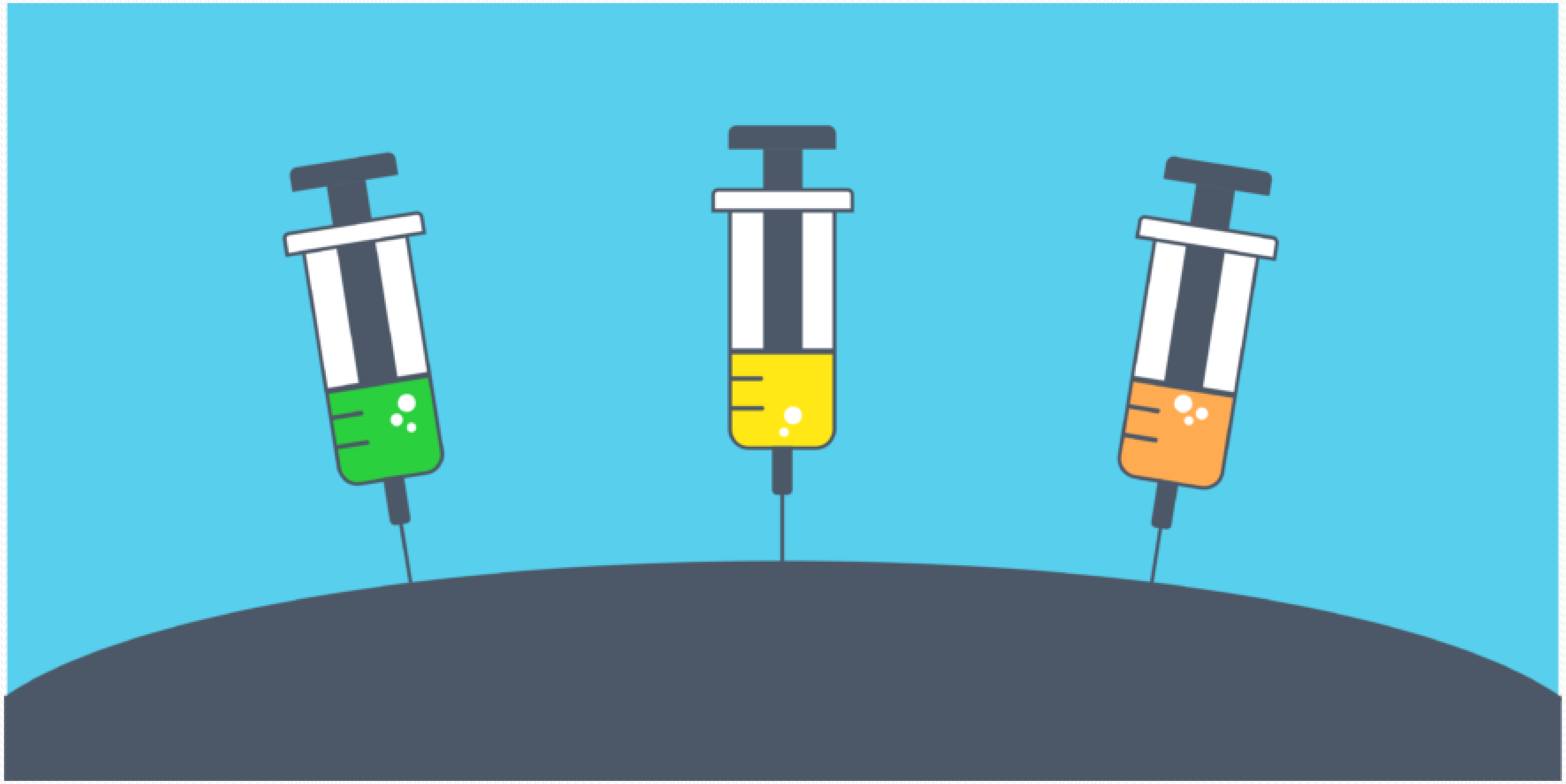
alphapeeler@icloud.com

pinterest.com/alphapeeler

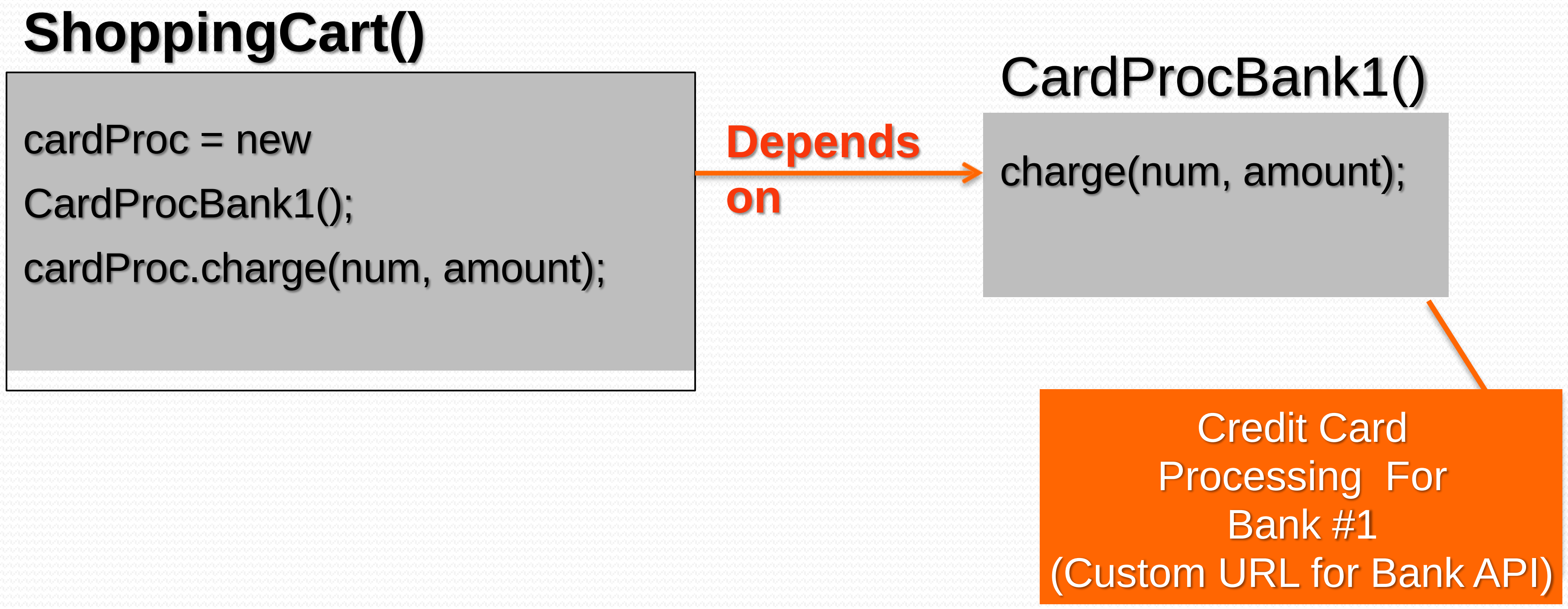
www.youtube.com/user/AlphaPeeler

Dependency Injection Pattern

Design pattern that implements *inversion of control* for resolving dependencies



“Regular” Control



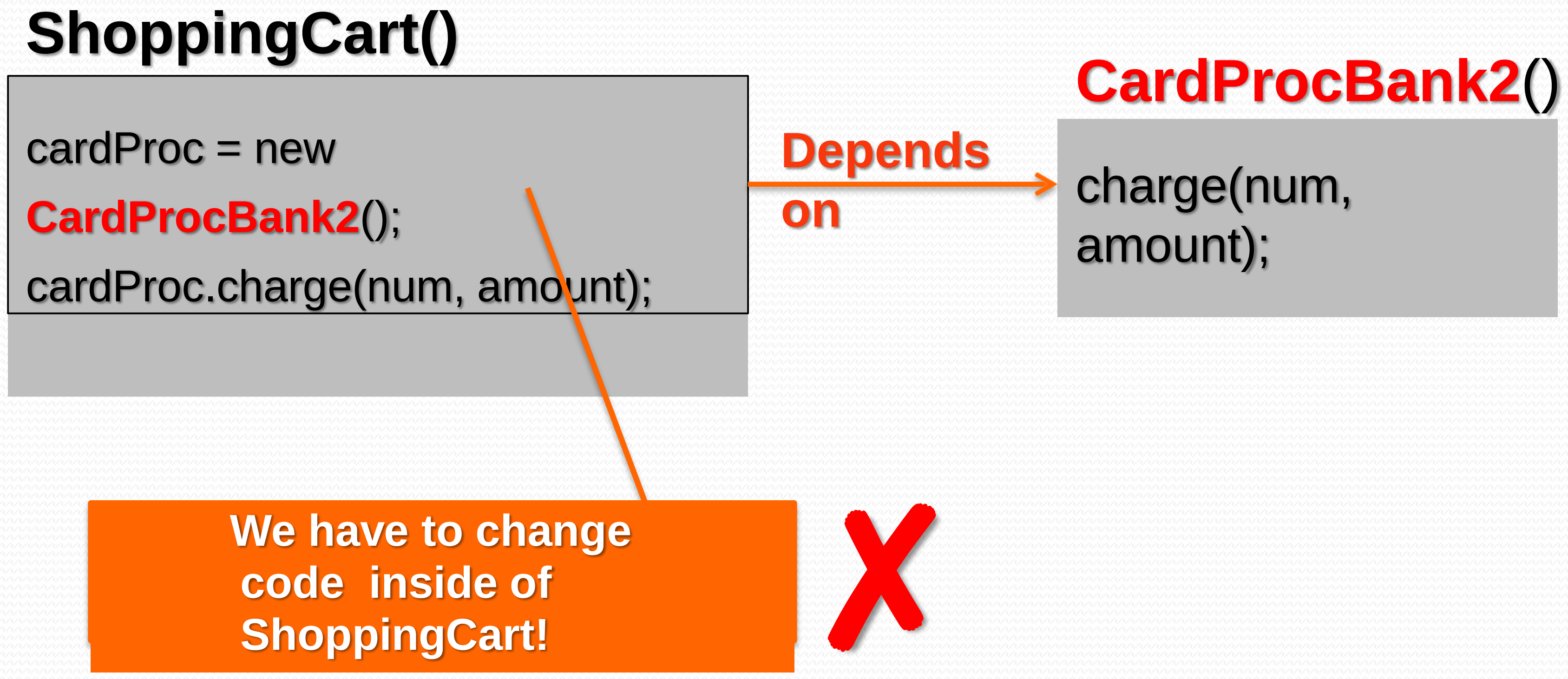
ShoppingCart()

```
cardProc = new
CardProcBank1();
cardProc.charge(num, amount);
```

CardProcBank1()

```
charge(num, amount);
```

“Regular” Control



```
cardProc = new
CardProcBank2();
cardProc.charge(num, amount);
```

```
charge(num,
amount);
```

We have to change code inside of ShoppingCart!

✗

Inversion of Control (IoC)

ShoppingCart(cardProc)

```
cardProc.charge(num, amount);
```

CardProcBank

```
charge(num, amount);
```

CardProcBank

```
charge(num, amount);
```

System

```
cardProc = new
CardProcBank1();
ShoppingCart(cardProc);
```

If we need a different bank for card processing, **ShoppingCart code will not change**

✓

Java Dependency Injection

- **Java Dependency Injection** design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement **dependency injection in java** to move the dependency resolution from compile-time to runtime.

Dependency of classes

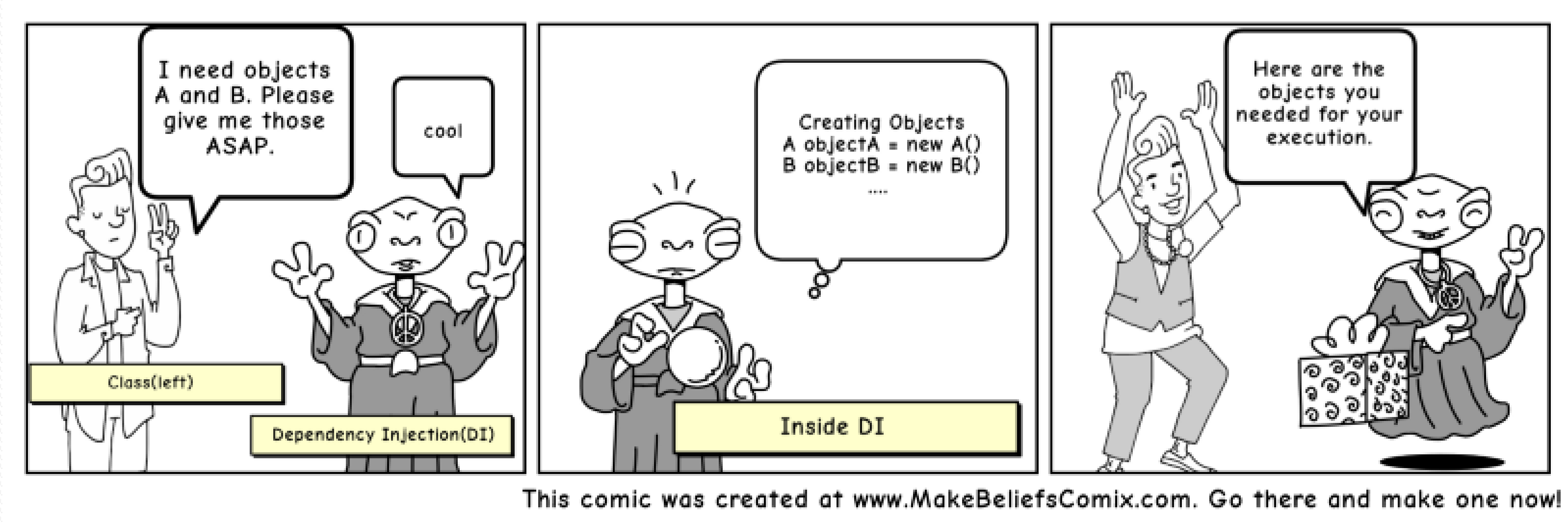
- So before getting to **dependency injections**, first let's understand what a dependency in programming means.

- When class A uses some functionality of class B, then its said that class A has a dependency of class B.

- In Java, before we can use methods of other classes, we first need to create the object of that class (i.e. class A needs to create an instance of class B).

dependency injection

- **So, transferring the task of creating the object to someone else and directly using the dependency is called dependency injection.**


I need objects A and B. Please give me those ASAP.
cool
Class(left)
Dependency Injection(DI)
Creating Objects
A objectA = new A()
B objectB = new B()
....
Inside DI
Here are the objects you needed for your execution.
This comic was created at www.MakeBeliefsComix.com. Go there and make one now!

Dependency injection

- Dependency injection is a programming technique that makes a class independent of its dependencies. It achieves that by decoupling the usage of an object from its creation. This helps you to follow SOLID’s **dependency inversion** and **single responsibility principles.**

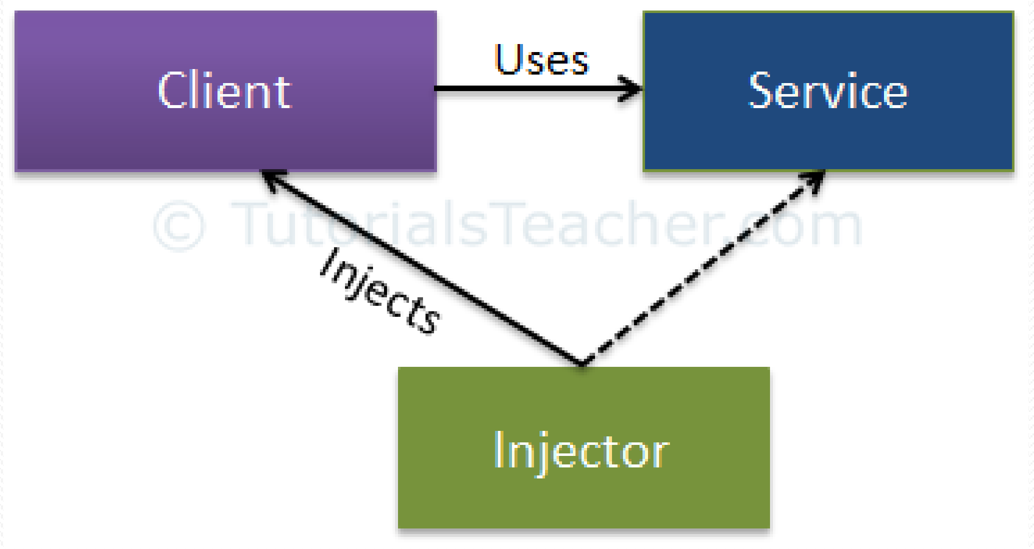
Libraries and Frameworks that implement DI

- AngularJS
- Spring (Java)
- Google Guice (Java)
- Dagger (Java and Android)
- Castle Windsor (.NET)
- Unity(.NET)

3 types of classes + interface

- The Dependency Injection pattern involves 3 types of classes.
- **Client Class:** The client class (dependent class) is a class which depends on the service class
- **Service Class:** The service class (dependency) is a class that provides service to the client class.
- **Injector Class:** The injector class injects the service class object into the client class.
- An **interface** that's used by the client and implemented by the service.

Dependency injection



Types of Dependency Injection

- The injector class injects dependencies broadly in three ways: through a constructor, through a property, or through a method.
- **Constructor Injection:** In the constructor injection, the injector supplies the service (dependency) through the client class constructor.
- **Property Injection:** In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.
- **Method Injection:** In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Ex1: The Leagacy App

```
public class EmailService {
public void sendEmail(String message, String receiver){
System.out.println("Email sent to "+receiver+ " with
Message="+message);
}}
```

```
public class MyApplication {
private EmailService email = new EmailService();
public void processMessages(String msg, String rec){
//do some msg validation, manipulation logic etc
this.email.sendEmail(msg, rec);
} }
public class MyLegacyTest {
public static void main(String[] args) {
// TODO Auto-generated method stub
MyApplication app = new MyApplication();
app.processMessages("Hello Sir A. Rahman",
"abdulrahman@nu.edu.pk");
}}
```

Output:

```
Email sent to
abdulrahman@nu.edu.pk with
Message=Hello Sir A. Rahman
```

The Leagacy App - Problems

- MyApplication class is responsible to initialize the email service and then use it. (hard-coded dependency.)
- If we want to switch to some other advanced email service in the future, it will require code changes in MyApplication class.

- If we want to extend our application to provide an additional messaging feature, such as SMS or Facebook message then we would need to write another application for that

Ex1: The Leagacy App – Bad Solution

- One can argue that we can remove the email service instance creation from MyApplication class by having a constructor that requires email service as an argument.

```
public class MyApplication {
private EmailService email = null;
public MyApplication(EmailService svc){
this.email=svc;
}
public void processMessages(String msg, String rec){
//do some msg validation, manipulation logic etc
this.email.sendEmail(msg, rec);
}}
```

- But in this case, we are asking client applications or test classes to initializing the email service that is not a good design decision.

Ex1: DIP Solution

- **Dependency Injection – Service Components**

```
public interface MessageService {
void sendMessage(String msg, String rec);
}

public class SMSServiceImpl implements MessageService {
@Override
public void sendMessage(String msg, String rec) {
//logic to send SMS
System.out.println("SMS sent to "+rec+ " with Message="+msg);
}}

public class EmailServiceImpl implements MessageService {
@Override
public void sendMessage(String msg, String rec) {
//logic to send email
System.out.println("Email sent to "+rec+ " with
Message="+msg);
} }
```

DIP – Service Consumer

```
public interface Consumer {
void processMessages(String msg, String rec);
}

public class MyDIApplication implements Consumer {
private MessageService service;
public MyDIApplication(MessageService svc){
this.service=svc;
}
@Override
public void processMessages(String msg, String rec){
this.service.sendMessage(msg, rec);
}}
```

DIP – Injectors Classes

```
public interface MessageServiceInjector {
public Consumer getConsumer();
}

public class EmailServiceInjector implements
MessageServiceInjector {
@Override
public Consumer getConsumer() {
return new MyDIApplication(new EmailServiceImpl());
}}

public class SMSServiceInjector implements
MessageServiceInjector {
@Override
public Consumer getConsumer() {
return new MyDIApplication(new SMSServiceImpl());
} }
```

Client App

- how our client applications will use the application with a simple program.

```
public class MyMessageDITest {
public static void main(String[] args) {
String msg = "Hi A. Rahman v3";
String email = "abdulrahman@nu.edu.pk";
String phone = "4088888888";
MessageServiceInjector injector = null;
Consumer app = null;

//Send email
injector = new EmailServiceInjector();
app = injector.getConsumer();
app.processMessages(msg, email);

//Send SMS
injector = new SMSServiceInjector();
app = injector.getConsumer();
app.processMessages(msg, phone);
} }
```

```
OutPut:
Email sent to
abdulrahman@nu.edu.pk with
Message=Hi A. Rahman v3
SMS sent to 4088888888 with
Message=Hi A. Rahman v3
```

```
angular.module('myApp', [])
.controller('MyCtrl', function ($scope) {
  $scope.name = "Yaakov";
});
```

Where did $scope come from?