package com.javacreed.examples.di.part1;

public class Email {

public void sendEmail(String subject, String message){

// Send the email

}

}

package com.javacreed.examples.di.part1;

public class Person {

private Email email = new Email();

public void greetFriend(){

email.sendEmail("Hello", "Hello my friend :)");

}

}

Person, and this class needs to send a message. The Person class requires the aid of some other class, Email, in order to send a message.

Let say we have a new and better version of email class, FastEmail, in order for us to use it, we need to go in each and every class that depends on the Email class, such as thePerson class, and replace it with the new version.

* Another developer needs to use the Person class, but would like to use a different notification/message system. This cannot be achieved with the current version of the Person class as it is hardwired to the Email class. What generally happens is that the other developer duplicates the Person class and modifies it as he/she needs. The projects ends up with two versions of the Person class.

we first create an interface,MessageService, and make the Person class using this interface instead. This removes the dependency that the Person class has on the Email and replaces it with an abstract message delivery interface

We start by defining the MessageService interface that includes one method, sendMessage(String subject, String message). For simplicity we assume that no exceptions are thrown.

package com.javacreed.examples.di.part2;

public interface MessageService {

void sendMessage(String subject, String message);

}

In the list of limitations we mentioned four possible methods of sending a message: email, fast email, SMS and tweet. Let’s create four classes that handle each message delivery method and have all these classes implement the interface created above.

package com.javacreed.examples.di.part2;

public class EmailService implements MessageService {

@Override

public void sendMessage(String subject, String message){

System.out.printf("Email: %s, %s%n", subject, message);

}

}

package com.javacreed.examples.di.part2;

public class FastEmailService implements MessageService {

@Override

public void sendMessage(String subject, String message){

System.out.printf("Fast Email: %s, %s%n", subject, message);

}

}

package com.javacreed.examples.di.part2;

public class SmsService implements MessageService {

@Override

public void sendMessage(String subject, String message){

System.out.printf("SMS: %s, %s%n", subject, message);

}

}

package com.javacreed.examples.di.part2;

public class TweetService implements MessageService {

@Override

public void sendMessage(String subject, String message){

System.out.printf("Tweet: %s, %s%n", subject, message);

}

}

Finally, instead of using classes, we use interfaces. In the Person class, we replace the Email field with the MessageService service interface as highlighted below.

public class Person {

private MessageService messageService;

public Person(MessageService messageService){

this.messageService = messageService;

}

public void greetFriend(){

messageService.sendMessage("Hello", "Hello my friend :)");

}

}

Note that the Person class is not initialising the message service but it is expecting it as a parameter of its constructor. This is a key element in the design. It improves **modularity, extendibility and testing**.

Dependency injection can help us initializing objects and provide these objects all the necessary resources (ingredients).