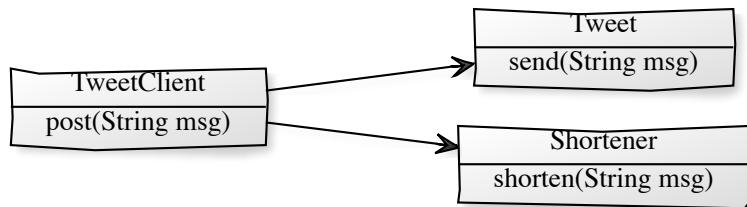


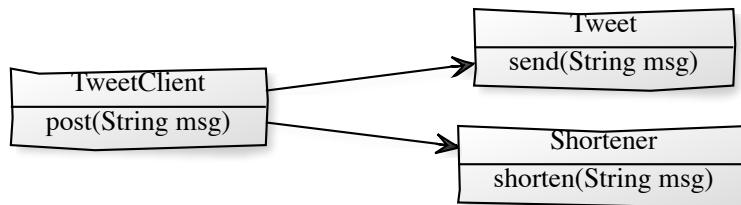
# Injection de dépendances

Frederico Alvares

# Exemple : Messagerie twitter



# Exemple : Messagerie twitter



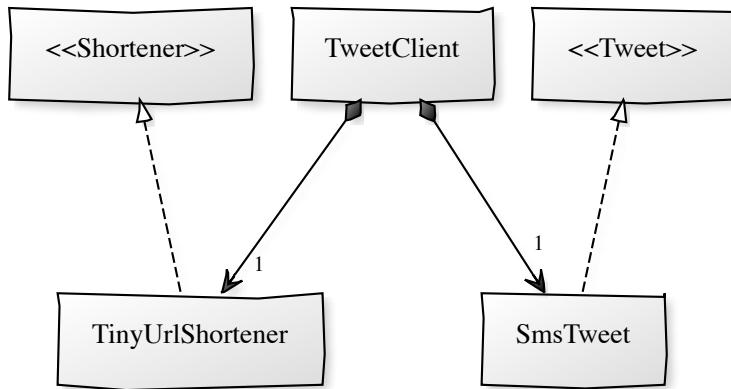
```
public void post(String msg){
    if (msg.length > 140){
        msg = shortener.shorten(msg);
    }

    if (msg.length <= 140){
        tweeter.send(msg);
    }
}
```



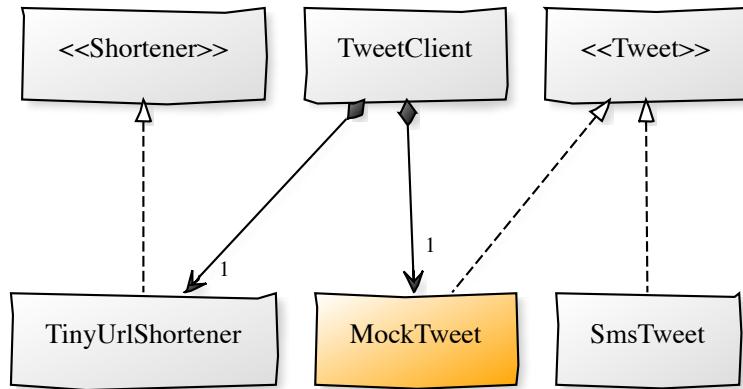
# Dépendances dans le constructeur

# Dépendances dans le constructeur



- Implémentez les classes indiquées dans le diagramme
- Implémentez une classe de teste pour tester votre implémentation
- Quels sont les inconvénients de l'approche par composition?

# Dépendances dans le constructeur

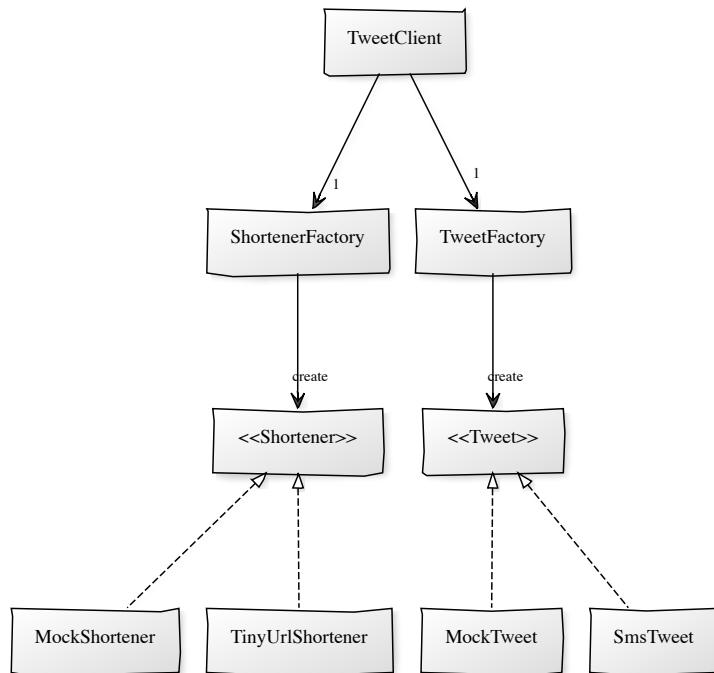


- Changement d'implémentation -> un changement dans la classe dépendante
- Manque de flexibilité
- Par exemple : pas très pratique pour les tests !

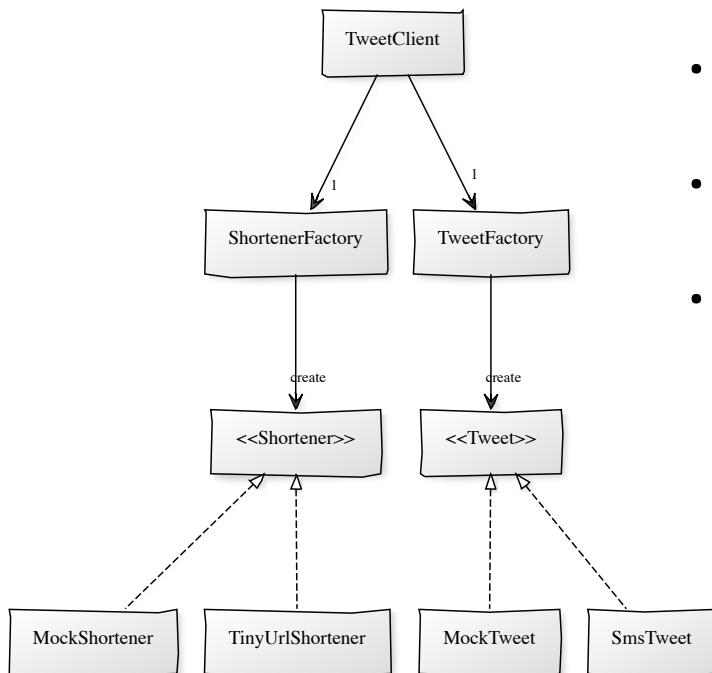


Dépendance via des fabriques statiques

# Dépendance via des fabriques statiques

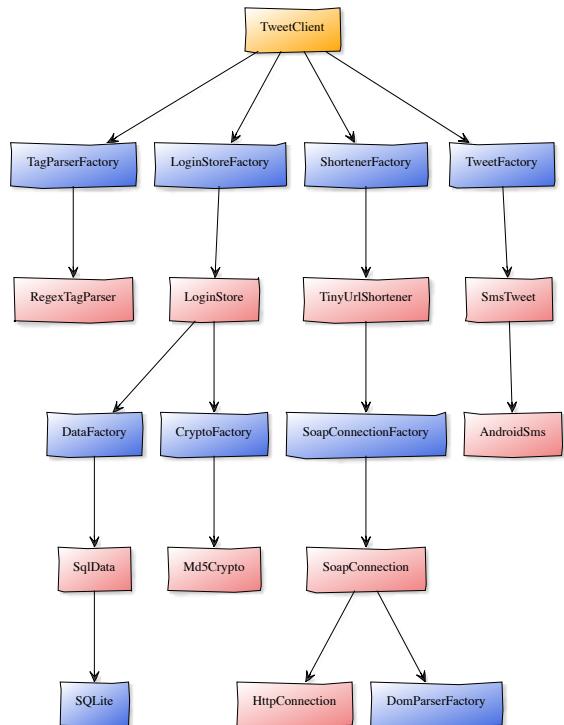


# Dépendance via des fabriques statiques

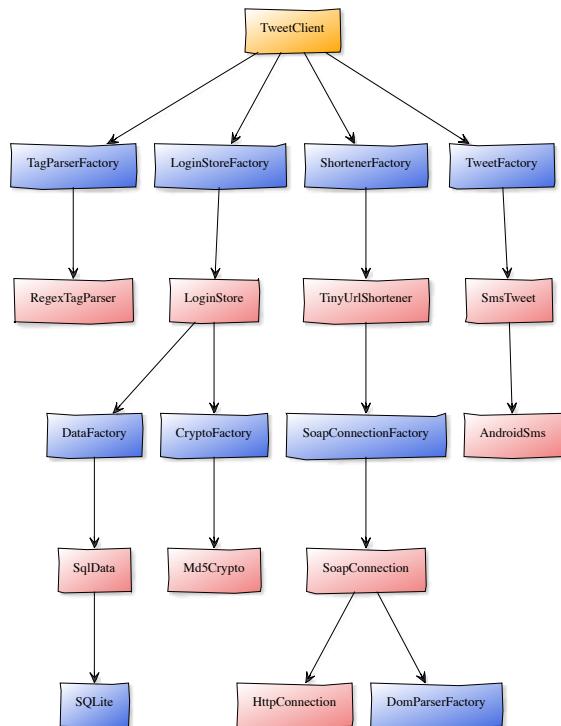


- Implémentez les classes indiquées dans le diagramme
- Implémentez une classe de teste pour tester votre implémentation
- Quels sont les avantages par rapport à l'approche par composition?
- Quels sont les inconvénients de cette approche?

# Dépendance via des fabriques statiques



# Dépendance via des fabriques statiques



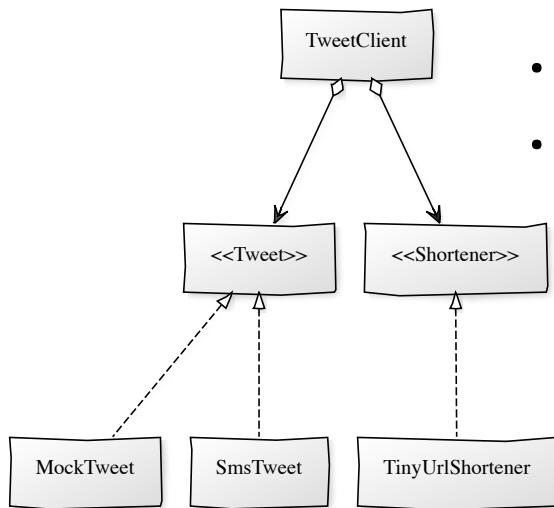
- On augmente le niveau d'abstraction mais le problème de dépendance reste le même.
- Testes.



# Injection de dépendances

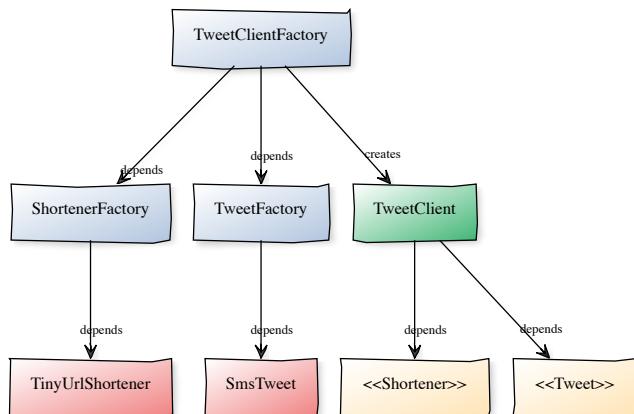
"Don't call us, we call you!"

# Injection de dépendances par agrégation



- Implémentez les classes indiquées dans le diagramme
- Implémentez une classe de teste pour tester votre implémentation
- Quels sont les avantages par rapport aux approches précédentes?

# Injection de dépendances par agrégation + fabriques



- Implémentez les classes indiquées dans le diagramme
- Implémentez une classe de teste pour tester votre implémentation
- Quels sont les avantages par rapport aux approches précédentes?
- Quels sont les inconvénients de cette approche?

# Coder les fabriques ?

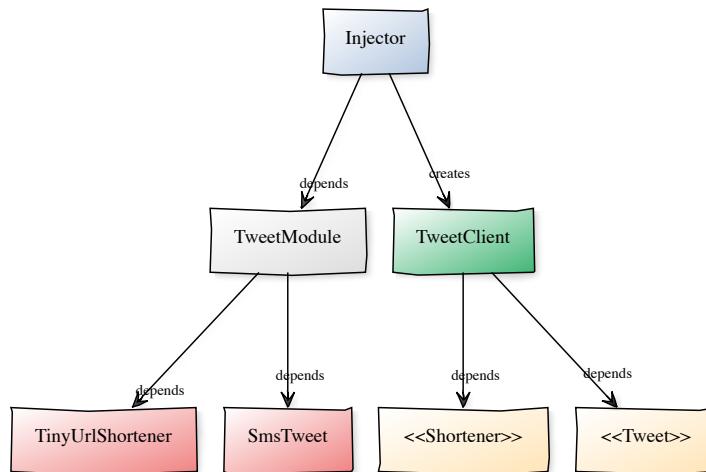


**What?  
No boilerplate!**



# Injection de dépendances avec Guice

# Injection de dépendances avec Guice



- Module : mapping interfaces -> classes implémentation
- Injector : joue le rôle d'une fabrique
  - Fournie par Guice
  - **on ne doit pas la coder/maintenir**

# Injection de dépendances avec Guice

- Module

```
import com.google.inject.AbstractModule;

public class TweetModule extends AbstractModule {
    @Override
    protected void configure() {
        bind(Twitter.class).to(SmsTwitter.class);
        bind(Shortener.class).to(TinyUrlShortener.class);
    }
}
```

# Injection de dépendances avec Guice

- Module

```
import com.google.inject.AbstractModule;

public class TweetModule extends AbstractModule {
    @Override
    protected void configure() {
        bind(Twitter.class).to(SmsTwitter.class);
        bind(Shortener.class).to(TinyUrlShortener.class);
    }
}
```

- Injector

```
Injector injector = Guice.createInjector(new TweetModule());
TweetClient client = injector.getInstance(TweetClient.class);
```

# Injection de dépendances avec Guice

- Classe dépendante

```
import com.google.inject.Inject;

public class TweetClient {
    ...
    @Inject
    public TwitterClient(Shortener shortener, Tweet twitter) {
        this.shortener = shortener;
        this.twitter = twitter;
    }
    ...
}
```

# A retenir : injection de dépendances et Guice

- Les objets viennent à vous
  - Au lieu de new et Factory
  - *Inversion of control (IoC)*
- Architecture plus modulaire / Modules réutilisables
- Fabriques déjà implémentées par Guice
- Moins de code inutile (factorisation)

