## Textual specification of the dependency types

5.1.1 Invocation of a method

Invocation of a method means that you call a method from a different class, which gives you a dependency to that class. For example, if the name of that method changes, or the amount of parameters the class calling that method won’t work anymore.

5.1.2 Invocation of a constructor

Invocation of a constructor means that you call a constructor from a different class, which gives you a dependency to that class. For example, if the amount of parameters of that constructor changes, the class trying to call that constructor won’t work anymore because.

5.2 Access of a property or field

Access of a property or field means that publicly/protected defined variables can be called from anywhere, which gives a dependency to all the classes calling this variable.

5.3 Extending a class/structure (struct)

The dependency from Extending a class/struct means that you are dependent on the class/struct you are extending. If this class/struct changes, the class extending them also inherits the changes from the parent class. This dependency type can be classified in the following sub-types:

* Extending an abstract class
* Extending an concrete class
* Extending a class of library; an example of this is the class JInternalFrame of the build-in Java library javax.swing

5.4 Implementing an interface

The dependency from implementing an interface means that you are dependent on the interface you are implementing. If the interface changes, the implementation of the interface will also change. For example if the interface gets different and/or more parameters the current implementation of the interface won’t work anymore.

5.5 Declaration

The dependency from using a declaration means that you are dependent on the field; variable or parameter declared, because using a class, struct or an enumeration. This dependency is mostly use in a combination with another dependency type.

5.6 Annotation of an attribute

The dependency from using an annotation of an attribute means that you are dependent on the existence of the annotation used. An annotation is a type of interface, but it is implemented differently. A normal interface you are only allowed to implement once in a class, but you are allowed to implement more than 1 annotation.

5.7 Throw an exception of class

The dependency from throwing an exception of a class means that parent class must have a link with the thrown exception which gives it a dependency to that class. For example when you define a custom exception and this exception is thrown, this means the parent class either has to throw the exception or catch it. In any case, the parent class is dependent on the custom exception. This is only the case when you have the throws keyword in your method declaration.

5.8 Typecast

The dependency created from typecasting means that you have to import the class or struct you wish to cast to. By doing so you automatically create a dependency with the class or struct you are currently typecasting.

## Examples of the dependency types

***5.1.1***

**public** **class** ASocialNetworkInfo {

**public** **static** String getInfo(){

**return** "The non-social information";

}

}

**public** **class** FeaturedMessages {

**public** FeaturedMessages(){

System.*out*.println(ASocialNetworkInfo.*getInfo*());

}

}

***5.1.2***

**public** **class** SocialNetwork {

**public** SocialNetwork(){

}

}

**public** **class** Account {

**public** Account(){

**new** SocialNetwork();

}

}

***5.2***

**public** **class** SocialNetworkInfo {

**public** **static** String *message* = "The social information";

}

**public** **class** Following {

**public** Following(){

System.*out*.println(SocialNetworkInfo.*message*);

}

}

***5.3***

**public** **abstract** **class** ASocialMedia {

}

**public** **class** Groups **extends** ASocialMedia {

}

***5.4***

**public** **interface** IASocialMedia {

}

**public** **class** IdenticaList **implements** IASocialMedia {

}

***5.5***

**public** **class** SocialNetworkInfo {

**public** Timeline time;

}

***5.6***

**public** **@interface** SocialTagAnnotation {

**public** String[] tags();

}

@SocialTagAnnotation(tags = { "three","different","tags" })

**public** **class** Trends {

}

***5.7***

**public** **class** ASocialMediaException **extends** Exception{

**public** ASocialMediaException(String message){

**super**(message);

}

}

**public** **class** PublicMessage {

**public** PublicMessage(){

**try** {

getMessages();

} **catch** (ASocialMediaException e) {

e.printStackTrace();

}

}

**public** String[] getMessages() **throws** ASocialMediaException{

**throw** **new** ASocialMediaException("No Messages");

}

}

***5.8***

**public** **class** SocialNetwork {

**private** String type = "default";

**public** String getSocialNetworkType(){

**return** this.type;

}

}

**public** **class** Followers {

**private** Object socialnetwork;

**public** Followers(){

System.*out*.println(((SocialNetwork)socialnetwork).getSocialNetworkType());

}

}