TND002 Object-oriented programming

Lecture 9: Concrete superclasses 3

Mark Eric Dieckmann,
MIT division,
ITN, Linköping University.

February 8, 2023

Outline of the lecture

- We discuss final methods and classes.
- Inner classes are introduced.
- We discuss how inner classes are inherited.

Final methods

```
public class Superclass {
protected final double variable = 1.0;
public double myMethod(){ return 1.0;} }
```

Declaring a variable as final turns it into a constant.

We can inherit the variables and methods of the superclass with

```
public class Subclass extends Superclass {
public double variable = 2.0; // overrides the inherited constant.
public double myMethod(){return 2.0;}}
```

Declaring *myMethod()* as final: *public final double myMethod(){}* blocks its overriding in the subclass. We get an error.

Final classes

Classes declared *final* cannot be a superclass.

```
public final class Base { // Code }
```

```
public class Derived extends Base { // Code}
```

This combination gives an error.

Examples for *final* classes are **String** and **Math**.

final in a class declaration prevents you from accessing or overriding its methods.

Declaring methods and classes as *final* can increase code security.

Introducing inner classes

Inner classes provide another way to structure code and to enhance security.

We illustrate how to use inner classes with a slimmed down version of the code in lab 2:

Word is only used by **Dictionary**.

```
public class Word {
private String theWord;
public Word(String arg) {theWord = arg;}
public String getWord() {return theWord;}
}
```

Picking a suitable inner class

Our slimmed-down class **Dictionary** is

```
import java.util.ArrayList;
public class Dictionary {
private ArrayList<Word> theList;
public Dictionary() {theList = new ArrayList<Word>();}
public void addWord(String arg) {theList.add(new Word(arg));}
public int listSize() {return theList.size();}
}
```

We can make **Word** an inner class of **Dictionary**.

It will work as if it was a separate class in the same package.

Making Word an inner class of Dictionary

```
import java.util.ArrayList;
public class Dictionary { // outer class
private ArrayList<Word> theList;
public Dictionary() {theList = new ArrayList<Word>();}
public void addWord(String arg) {theList.add(new Word(arg));}
public int listSize() {return theList.size();}
public String getWord(int i) {return theList.get(i).getWord();}

public class Word
{ // inner class
private String theWord;
public Word(String arg) {theWord = arg;}
public String getWord() {return theWord;}}}
```

How about static inner classes

Keeping **Word** as a separate class allowed us to instantiate it independently of **Dictionary**.

Declaring the inner class as *public class Word* means that each instance **Dictionary** carries its own copy of **Word**'s code.

We can only create an instance of **Word** if we already have an instance of **Dictionary**.

Declaring variables *static* allows us to use them before we create instances of the class.

The same is true for inner classes. Declare it static and you can access it even without having an instance of **Dictionary**.

Static and non-static inner classes

```
import java.util.ArrayList;
public class Dictionary { // outer class
private ArrayList<Word> theList;
private ArrayList<WordS> theListS;
public Dictionary() {theList = new ArrayList<Word>();
theListS = new ArrayList<WordS>();}
public void addWord(String arg) {theList.add(new Word(arg));}
public void addWordS(String arg) {theListS.add(new WordS(arg));}
public String listSize() {return "Instance: " + theList.size() + " and
Class: " + theListS.size();}
public class Word {private String theWord;
public Word(String arg) {theWord = arg;}
public String getWord() {return theWord;}}
public static class WordS {private String theWord;
public WordS(String arg) {theWord = arg;}
public String getWord() {return theWord;}} }
```

Accessing the inner class from the method *main(..)*

We can access the inner class like that

```
public class MainClass {
public static void main(String[] args) {
Dictionary.WordS wS = new Dictionary.WordS("class Hi");
System.out.println(wS.getWord());
Dictionary dict = new Dictionary();
Dictionary.Word w = dict.new Word("instance Hi");
System.out.println(w.getWord());
dict.addWord("Hi");dict.addWordS("Hi");
System.out.println(dict.listSize());}}
```

Console output:

```
class Hi
instance Hi
Instance: 1 and class: 1
```

Testing a *private static* inner class

```
public class DictionaryS {
private WordS theWordS = new WordS("there!");
public String returnVariables()
{return WordS.staticWord + theWordS.theWord;}
public String returnMethods()
{return WordS.getStaticWord() + theWordS.getWord();}
private static class WordS{ private String theWord;
private static String staticWord = "Hi ";
public WordS(String arg) {theWord = arg;}
public String getWord() {return theWord;}
public static String getStaticWord() {return staticWord;}}}
```

```
public class MainClass { // writes "Hi there!" twice to the console.
public static void main(String[] args) {
DictionaryS dictS = new DictionaryS();
System.out.println(dictS.returnVariables());
System.out.println(dictS.returnMethods()); }}
```

Testing a *private* inner class

```
public class DictionaryI {
private WordI theWordI = new WordI("there!");
public String returnVariables()
{return WordI.staticWord + theWordI.theWord;}
public String returnMethods()
{return WordI.getStaticWord() + theWordI.getWord();}
private class WordI { private String theWord;
private static String staticWord = "Hi ";
public WordI(String arg) {theWord = arg;}
public String getWord() {return theWord;}
public static String getStaticWord() {return staticWord;} } }
```

```
public class MainClass { // writes "Hi there!" twice to the console.
public static void main(String[] args) {
DictionaryI dictI = new DictionaryI();
System.out.println(dictI.returnVariables());
System.out.println(dictI.returnMethods());}}
```

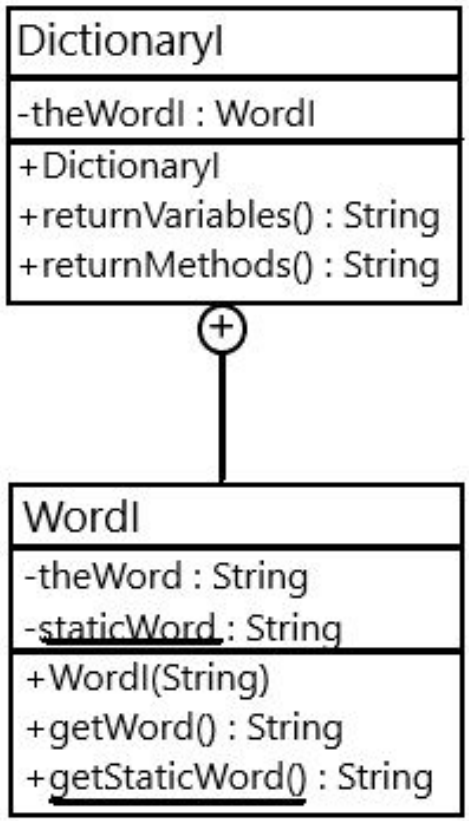
Class diagram of inner classes

We show the class diagram of **DictionaryI** with the non-static inner class **WordI**.

Inner and outer classes get their own box.

The line connected with the encircled plus states that **WordI** is an inner class.

It resembles an inheritance diagram for a subclass **WordI**.



Inner classes are however not sub- or superclasses.

Advantages of using an inner class

We can integrate a class, which is only used by one class, into the latter class (making it the outer class).

If we change the access modifier of an inner class from *public* to *private*, it cannot be used outside the outer class.

Its variables and methods are accessible inside the outer class.

Private inner classes can not be inherited.

We could accomplish the same by setting the access modifiers of methods to private (accessible only inside the class) but inner classes provide a better structure.

We can have a separate class **Word** (outside **Dictionary**) in our package and use it like before.

Summary

We have discussed in this lecture

- final classes.
- inner classes.
- inheritance of inner classes.