How to Think Like a Java Programmer

Dr Heinz M. Kabutz

Last Updated 2022-06-07

Copyright Notice

Programming is a team sport - VCS

- **Computer systems have millions of lines of code**
  - Programming is a team sport
  - Not practical for a single programmer to create everything
  - We all have different strengths and weaknesses
    - Database, networking, architecture, design, UI
- **We use Version Control Systems (VCS)**
  - Keeps track of all our changes and different versions of code
  - We will use Git for this course
    - Maybe create your own GitHub account if you don't have one

Who Am I?

- **Dr Heinz Kabutz**
  - Born in Cape Town, South Africa, now lives on Crete
    - Founder of JCrete - See https://www.jcrete.org
  - Created The Java Specialists' Newsletter
    - https://www.javaspecialists.eu/archive/
  - One of the first Java Champions
    - http://javachampions.org

Questions

- **There are some stupid questions**
  - They are the ones we did not ask
  - Once asked, they are no longer stupid
- **The more you ask, the more we all learn**
  - And the more fun we will have today

How to ask Questions on ON24?

- **We can ask questions in the group chat**
  - Let's try it now to make sure that it works for you
    - How warm is it where you are at the moment? In C or F

Fahrenheit (F)	Celsius (C)
-40	-40
0	-18
32	0
50	10
77	25
86	30
95	35
104	40

0. Setup

Oracle JDK

- **Oracle owns the Java brand**
  - Oracle JDK
    - www.oracle.com/java/technologies/javase-downloads.html
    - Running this in production might be an issue
      - Speak to your lawyer
  - Oracle OpenJDK
    - https://jdk.java.net
    - Doesn't have such a nice installer
      - But otherwise, exactly the same as the Oracle JDK
- **Eclipse Adoptium Temurin OpenJDK**
  - https://adoptium.net/

Java Versions

- **Java 17 released in September 2021**
  - Every 6 months a "feature release"
- **Java 11 released in September 2018**
- **Java 8 released in March 2014**
- **Java 7 released in July 2011**
- **Even in 2022, a lot of companies still on Java 8**

Integrated Development Environments

- **Java is too complex to write in Notepad**
  - We need tool help, from an IDE
    - Autocompletion, syntax highlighting, etc.
- **Most popular ones are**
  - IntelliJ IDEA
  - Eclipse IDE
  - Visual Studio Code
  - Apache NetBeans
- **We will use IntelliJ IDEA Community Edition**
  - www.jetbrains.com/idea/download

IntelliJ IDEA Community Edition

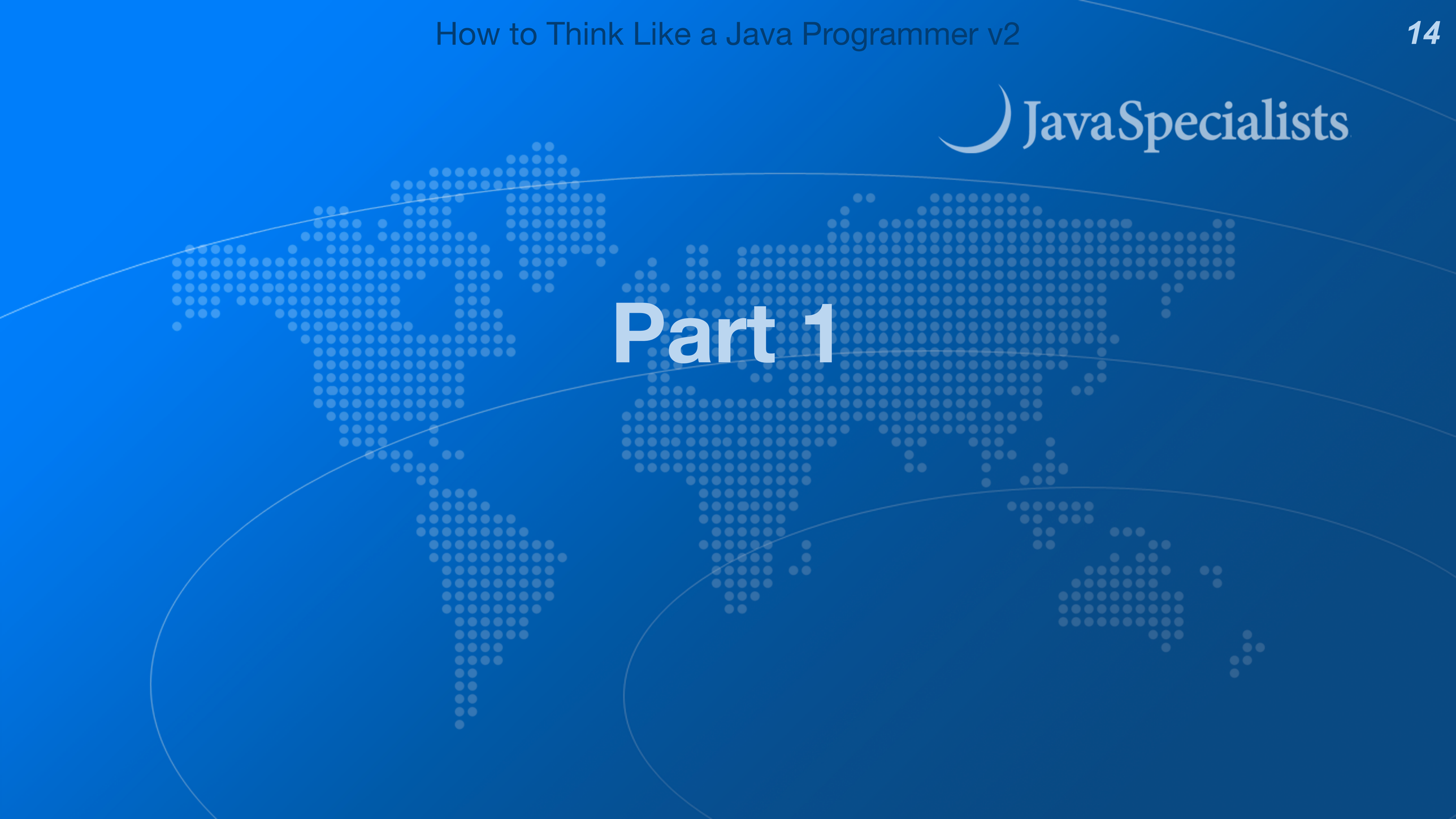
- **All you need to choose is your color scheme**
  - Darcula is the most popular at 61.7%

Loading our Project in IntelliJ

- **Go to File → New → Project from Version Control ...**
- **Enter the Repository URL**
  - https://user:password@javaspecialists.eu/git/safari/Juppies-####.git
    - (Not browseable)
- **Select a destination directory on your disk**
- **Click "Clone"**
- **Click "Trust Project"**

Running AnagramGame with IntelliJ

- **Configure which JDK to use (JDK 17 or later)**
  - File ➔ Project Structure...
- **Set up toolbar and autoscrolling**
- **Press Shift Shift and type Anagrams.java and Enter**
- **Now press Control + Shift + F10 to run**
  - Once you have run it once, press Shift + F10 to run again



Part 1

AnagramGame Project Structure

- **Our project is arranged in packages like directories**
  - All packages begin with my domain name eu.javaspecialists
    - Then courses.juppies.anagrams
      - Then lib and ui

WordLibrary

```java
/** Word library defining logic for the Anagram Game. */
public abstract class WordLibrary {
    public abstract String getWord(int idx);
    public abstract String getScrambledWord(int idx);
    public abstract int getSize();
    public boolean isCorrect(int idx, String userGuess) {
        return userGuess.equals(getWord(idx));
    }
}
```

- Comments = Help to explain how code works
- Public = Anyone can see it
- Abstract = Abstract concept such as love, success, freedom
- Strongly typed language
- Comparing with == versus .equals()

Swing User Interface (UI)

- **Cross platform graphical user interface**
- **Used in many applications**
- **Alternatives**
  - Eclipse RCP
    - Has a more native look
  - JavaFX
    - Richer graphics
    - See openjfx.io

Demo of New Word Button

- % is remainder

```
wordIdx = (wordIdx + 1) % wordLibrary.getSize();
scrambledWord.setText(wordLibrary.getScrambledWord(wordIdx));
```

Demo of Guess Button

```
if (wordLibrary.isCorrect(wordIdx, yourGuess.getText())) {
    feedbackLabel.setText("Correct! Try a new word!");
    getRootPane().setDefaultButton(newWord);
} else {
    feedbackLabel.setText("Incorrect! Try again!");
    yourGuess.setText("");
}
yourGuess.requestFocusInWindow();
```

StaticWordLibrary

```
public final class StaticWordLibrary extends WordLibrary {
    private static final String[] WORD_LIST = {
            "dolphin", "shark", "octopus", "seahorse",
            "whale", "penguin", "seal",
    };
    private static final String[] SCRAMBLED_WORD_LIST = {
            "odplnhi", "kahrs", "cootspu", "sehosear",
            "eahlw", "epgnnui", "lsea",
    };
    public String getWord(int idx) {
        return WORD_LIST[idx];
    }
    public String getScrambledWord(int idx) {
        return SCRAMBLED_WORD_LIST[idx];
    }
    public int getSize() { return WORD_LIST.length; }
}
```

Changing the StaticWordLibrary Size

```
public int getSize() {
    return 5;
}
```

```
public int getSize() {
    return 10;
}
```

Adding Our Own Words to the Library

- **Let's add some words to the StaticWordLibrary**
  - "beer", "chocolate", "coffee",
  - "ebre", "checalota", "eeffoc",
- **Play and see if we can guess them**

WordLibraryTest With JUnit

- **In programming we always want to test our work**
  - It is too easy to make silly mistakes
- **The most popular testing tool in Java is JUnit**
  - Let's look at the WordLibraryTest

Java's For Loop

```
for (int i = 0; i < wordLibrary.getSize(); i++) {
    // ...
}
```

```
int i = 0;
while (i < wordLibrary.getSize()) {
    // ...
    i++;
}
```

Testing That Words Are Anagrams

```
assertTrue(isAnagram(clearWord, scrambledWord),
        "Scrambled word \"" + scrambledWord +
                "\" at index: " + i +
                " does not represent the word \"" +
                clearWord + "\"");
```

Running JUnit

- **Shows the spelling mistake immediately**
  - Best to run the unit tests before we run the program

isAnagram() Method

```
private boolean isAnagram(String clearWord,
                          String scrambledWord) {
    char[] clearArray = clearWord.toCharArray();
    char[] scrambledArray = scrambledWord.toCharArray();
    Arrays.sort(clearArray);
    Arrays.sort(scrambledArray);
    return Arrays.equals(clearArray, scrambledArray);
}
```

Fixing StaticWordLibrary

- Fix and run the test again
- "Keep it green to keep it clean"

WordLibraries Facade

- **Facades usually are written in plural form**
  - Note the CamelCase spelling

```
public final class WordLibraries {
    private WordLibraries() {
    }

    public static WordLibrary createDefaultWordLibrary()
        return new StaticWordLibrary();
    }
}
```

Find Usages

- **Right-click createDefaultWordLibrary() and "Find Usages"**
  - Used in Anagrams.java and WordLibraryTest.java

Exercise

- **Please add some words to the StaticWordLibrary**
  - No one is looking, so you can add some "naughty" words

Git: Create Branch

- **Branches allow us to develop in parallel**
  - Every developer has their own branch
- **Git → New Branch...**
  - Choose a branch name with your initials
    - Don't add more information than you want everyone to see!
- **Usually we create a branch for new features**
  - Once we are done, we can merge back into the main branch
    - Also called "master" or "trunk"

Commiting Changes

- **Commit your changes with Git → Commit...**
  - Add a good message detailing what you have changed
    - e.g. "Added three of my favourite things"
    - In real world we would include a change request number
- **The commit is only on your own disk at the moment**
  - We don't want to push our changes in a large group like this

JavaSpecialists

Part 2

Welcome back

- **Words always in the same order**
  - We will learn how to shuffle the words
  - And a bit about software design

Typical Maven Directory Structure

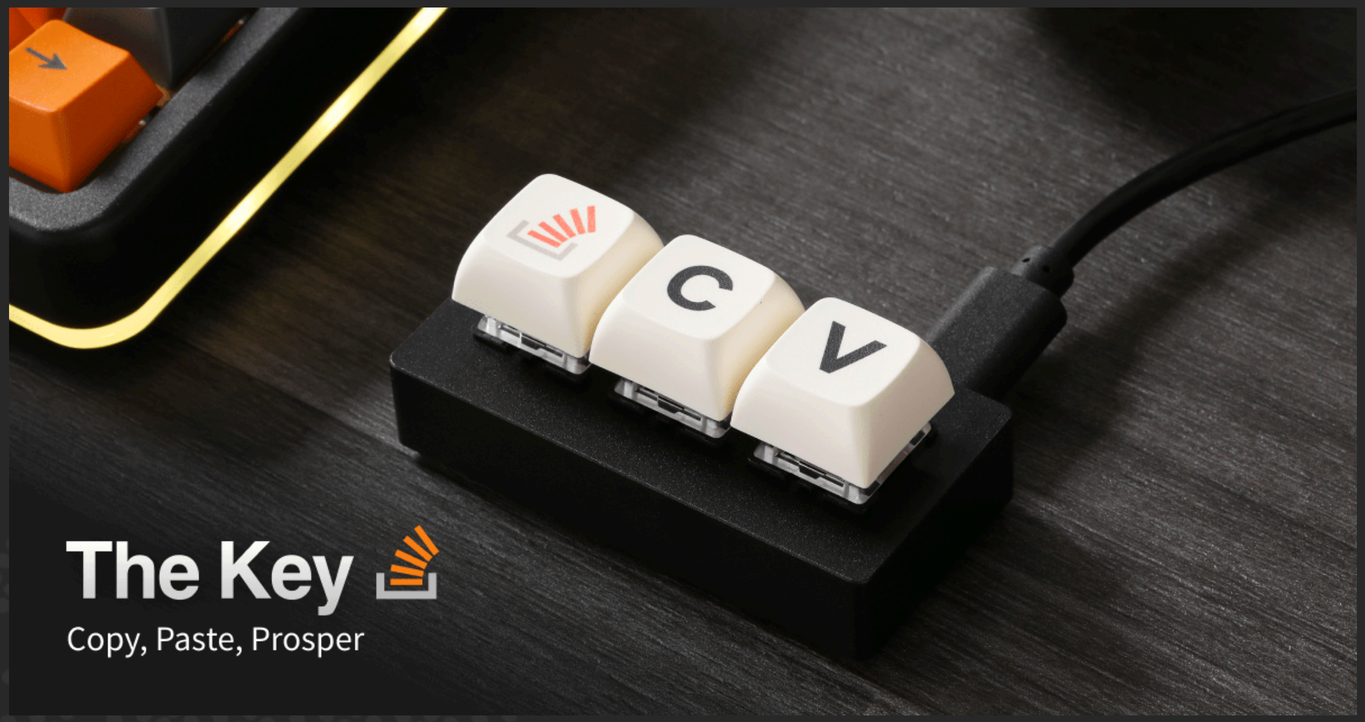
- **Directory structure is usually**
  - src/main/java - for our Java code
  - src/main/resources - for properties files, images, etc.
  - src/test/java - for our test code
  - src/test/resources - any resources for our tests
- **Packages are reflected in directory structure**
  - eu.javaspecialists.courses.juppies.anagrams.lib is located in eu/javaspecialists/courses/juppies/anagrams/lib

IntelliJ Magic Shortcuts

- See IntelliJ Wizardry with Heinz Kabutz Course
- Learn one new shortcut a day

Avoid Copy & Paste Programming

- **Copy & Paste coding is dangerous**
  - False sense of productivity
  - Bugs also get copied and pasted
  - Easy to violate license terms



Creating a ShuffledWordLibrary

- **Let's create a ShuffledWordLibrary**
  - (Note to self - slow down here)
  - Click on WordLibrary
  - Alt+Enter or Option+Enter ➔ Implement Abstract Class
    - Call it ShuffledWordLibrary
  - Type in "private final WordLibrary other;"
    - Alt+Enter or Option+Enter ➔ Add constructor parameter
  - Constructor has same name as the class, but no return type

Integer Array for Shuffled Indexes

- Type "private final int[] indexes;" in class
- Initialize indexes in constructor and set the values

```
public ShuffledWordLibrary(WordLibrary other) {
    this.other = other;
    indexes = new int[other.getSize()];
    for (int i = 0; i < indexes.length; i++) {
        indexes[i] = i;
    }
}
```

Delegate to the other WordLibrary

- Next we fill in the getWord(), getScrambledWord() and getSize() methods

```
@Override
public String getWord(int idx) {
    int newIdx = indexes[idx];
    return other.getWord(newIdx);
}
@Override
public String getScrambledWord(int idx) {
    int newIdx = indexes[idx];
    return other.getScrambledWord(newIdx);
}
@Override
public int getSize() {
    return other.getSize();
}
```

Trying out the "ShuffledWordLibrary"

- **Take small careful steps forward during coding**
- **In WordLibraries, change the code to**

```
public static WordLibrary createDefaultWordLibrary() {
    return new ShuffledWordLibrary(new StaticWordLibrary());
}
```

- **We run the Anagrams program to try it out**
  - Still the original order

Committing Our Changes to Git

- **Commit to Git often**
  - You can squash the commits if you have too many
- **It's a bit like playing a tricky computer game**
  - Save regularly in case you run out of ammo

Shuffling the Integer Array

- **Let's add a shuffle(int[] indexes) method**
  - Take last element, swap with a random element on the left
  - Take 2nd last element, swap again with a random on the left

```
private void shuffle(int[] indexes) {
    Random random = new Random(0);
    for (int i = indexes.length - 1; i > 0; i--) {
        int swap = random.nextInt(i + 1);
        if (swap != i) {
            int tmp = indexes[i];
            indexes[i] = indexes[swap];
            indexes[swap] = tmp;
        }
    }
}
```

Trying Out the Shuffled Anagram List

- Let's run it a few times ...

How can we test private methods?

- **Best to move the shuffle() method to another class**
  - And make the method public
  - Change the parameter to "values" instead of "indexes"
    - The name indexes is very specific to how we are using it

ArrayShuffler

- Let's create a new class ArrayShuffler

```
package eu.javaspecialists.courses.juppies.anagrams.util;

import java.util.Random;

public class ArrayShuffler {
    public void shuffle(int[] values) {
        Random random = new Random(0);
        for (int i = values.length - 1; i > 0; i--) {
            int swap = random.nextInt(i + 1);
            if (swap != i) {
                int tmp = values[i];
                values[i] = values[swap];
                values[swap] = tmp;
            }
        }
    }
}
```

Creating an ArrayShufflerTest

- We Alt+Enter on ArrayShuffler to make a JUnit5 test

```
@Test
public void testShuffle() {
    int[] indexes = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    ArrayShuffler shuffler = new ArrayShuffler();
    shuffler.shuffle(indexes);
    boolean different = false;
    for (int i = 0; i < indexes.length; i++) {
        if (i != indexes[i]) different = true;
    }
    if (!different)
        fail("Shuffling did not do anything.");
}
```

- Now we comment out the shuffle() call and try again

Printing the indexes array

- **Let's print the indexes array to the output**

```
System.out.println("indexes = " + indexes);
```

  - Produces this output: `indexes = [I@27082746`

- **For arrays, we need to do a bit more work**

```
System.out.println("indexes = " + Arrays.toString(indexes));
```

  - This is better: `indexes = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

- **When we run the code, we see this after shuffling**

```
[4, 8, 9, 6, 3, 5, 2, 1, 7, 0]
```

Extending the Tests

- Let's extends test to check for better randomness

```
@Test
public void testShuffle() {
    int[] indexes = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    String beforeShuffle = Arrays.toString(indexes);
    System.out.println("before shuffle indexes = " + beforeShuffle);
    ArrayShuffler instance = new ArrayShuffler();
    instance.shuffle(indexes);
    String afterShuffle = Arrays.toString(indexes);
    System.out.println("after shuffle indexes = " + afterShuffle);
    assertNotEquals("[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]",
        afterShuffle, "Shuffling did not do anything");
    assertNotEquals("[4, 8, 9, 6, 3, 5, 2, 1, 7, 0]",
        afterShuffle, "Shuffling was in a predictable sequence");
}
```

Fixing Random to be more random

- Simple fix, we take away the 0 seed in Random

```
public void shuffle(int[] values) {
    Random random = new Random();
    for (int i = values.length - 1; i > 0; i--) {
        int swap = random.nextInt(i + 1);
        if (swap != i) {
            int tmp = values[i];
            values[i] = values[swap];
            values[swap] = tmp;
        }
    }
}
```

- Let's run the test again!

The Goat, The Wolf and The Cabbage

- "How do we swap two int values without using a temporary variable?"
- We can do it with addition and subtraction
  - `x=x+y → y=x-y → x=x-y`
  - `x=4,y=7 → x=11,y=7 → x=11,y=4 → x=7,y=4`

```
values[i] += values[swap];
values[swap] -= values[i];
values[i] -= values[swap];
```

Testing the Integer Swapping Code

- **Let's run the game**
  - Oops - `ArrayIndexOutOfBoundsException`
- **We should add another test, such as**

```
Arrays.sort(indexes);
String sorted = Arrays.toString(indexes);
assertEquals("[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]",
        sorted, "Not same values as before shuffle");
```

Testing Before Running Application

- **Even with the bug, the application sometimes starts**
  - The first index might be within range
- **It would be better if all the tests were run each time**
  - In Project tool window, right-click Anagram / src / test / java and select "Run 'All Tests' "
  - Next select Anagrams.java, right-click and select "More Run/Debug" ➔ Modify Run Configuration...
    - Then Modify Options ➔ Add before launch task
      - Select "Run Another Configuration"
        - » And select "All in AnagramGame"

Fixing the Integer Swapping Code

- **Overconfidence leads to bugs**

```
public void shuffle(int[] values) {
    Random random = new Random();
    for (int i = values.length - 1; i > 0; i--) {
        int swap = random.nextInt(i + 1);
        if (swap != i) {
            values[i] += values[swap];
            values[swap] = values[i] - values[swap];
            values[i] -= values[swap];
        }
    }
}
```

Integer Swapping With XOR

- **We can also swap two ints using exclusive or XOR**

```
x ^= y;
y ^= x;
x ^= y;
```

- **Let's apply that to our code**

```
values[i] ^= values[swap];
values[swap] ^= values[i];
values[i] ^= values[swap];
```

- **Whilst this looks cool, rather don't do this**
  - Principle of least astonishment (POLA)

Value of Source Control Systems

- **Let's check in our XOR int swap code**
- **With each change as commit, we have a full history**
  - Sadly, even today, some smaller companies use USB sticks

Abstract Class vs Interface

- **An abstract class is a partial solution**
  - Leaving some details to be filled in by subclasses
  - The abstract class can have fields
  - A class can only extend a single class, abstract or not
- **An interface is a contract for subclasses to fulfil**
  - A class can implement as many interfaces as we want to
  - Can have "default" methods that depend on other methods
    - Also possible to have static or private methods, but that's rare
  - An interface cannot have fields, except for constants

Change WordLibrary an Interface

- It is easy to change the abstract class to an interface

```
public interface WordLibrary {
    String getWord(int idx);
    String getScrambledWord(int idx);
    int getSize();
    default boolean isCorrect(int idx, String userGuess) {
        return userGuess.equals(getWord(idx));
    }
}
public final class StaticWordLibrary implements WordLibrary
public class ShuffledWordLibrary implements WordLibrary
```

Commit

- **Again, let's commit to Git with a good message**
  - I will push my changes to the "kabutz" branch
    - But please don't push your changes
    - With 100+ students, this is likely to cause chaos
  - In a group project, you need to first pull others' changes

JavaSpecialists

Part 3

Welcome back

- **What have we seen so far**
  - Part 1: We looked at how the anagram game and Git worked
  - Part 2: We learned to shuffle the words and JUnit testing
- **Next up**
  - Letters in words were always scrambled the same way
    - "eber" ➔ "beer" and "odplnhi" ➔ "dolphin"
      - Gets too easy after a few times
  - We want to scramble the letters using different techniques
    - Similar to shuffling words in Part 2

ScrambledWordLibrary

- **Let's create a ScrambledWordLibrary**
  - Add `private final WordLibrary other;`
    - Also a constructor
  - Delegate getWord(), getScrambledWord(), getSize() to other

```
public abstract class ScrambledWordLibrary implements WordLibrary {
    // ...
    @Override
    public String getScrambledWord(int idx) {
        char[] letters = other.getScrambledWord(idx).toCharArray();
        scramble(letters);
        return new String(letters);
    }

    protected abstract void scramble(char[] letters);
}
```

SortedScrambledWordLibrary

- Our first implementation sorts the characters
- Alt+Enter → Implement Abstract Class

```
import java.util.Arrays;
public class SortedScrambledWordLibrary
        extends ScrambledWordLibrary {
    public SortedScrambledWordLibrary(WordLibrary other) {
        super(other);
    }
    @Override
    protected void scramble(char[] letters) {
        Arrays.sort(letters);
    }
}
public static WordLibrary createDefaultWordLibrary() {
    return new SortedScrambledWordLibrary(
            new ShuffledWordLibrary(new StaticWordLibrary()));
}
```

eginnpu, aehlw, ceeffo, aeehorss???

Perhaps we need a "Hint" button?

RandomScrambledWordLibrary

- **We copy the SortedScrambledWordLibrary**
  - Using ArrayShuffler's shuffle method

```
import eu.javaspecialists.courses.juppies.anagrams.util.*;

public class RandomScrambledWordLibrary
        extends ScrambledWordLibrary {
    public RandomScrambledWordLibrary(WordLibrary other) {
        super(other);
    }

    @Override
    protected void scramble(char[] letters) {
        ArrayShuffler shuffler = new ArrayShuffler();
        shuffler.shuffle(letters); // does not compile yet
    }
}
```

Adding a shuffle(char[]) method

- **We create this shuffle(char[]) in ArrayShuffler**
  - Exactly the same as for the int[]
  - Unfortunately in Java, int[] and char[] are totally different

```
public void shuffle(char[] values) {
    Random random = new Random();
    for (int i = values.length - 1; i > 0; i--) {
        int swap = random.nextInt(i + 1);
        if (swap != i) {
            values[i] ^= values[swap];
            values[swap] ^= values[i];
            values[i] ^= values[swap];
        }
    }
}
```

Testing the shuffle(char[]) method

```
@Test
public void testShuffleChars() {
    char[] letters = "hello world".toCharArray();
    ArrayShuffler shuffler = new ArrayShuffler();
    shuffler.shuffle(letters);
    String shuffle1 = new String(letters);

    letters = "hello world".toCharArray();
    shuffler.shuffle(letters);
    String shuffle2 = new String(letters);

    Arrays.sort(letters);
    String sorted = new String(letters);

    assertNotEquals("hello world", shuffle1);
    assertNotEquals(shuffle1, shuffle2);
    assertEquals(" dehllloorw", sorted);
}
```

Exercise

- **Improve our int[] shuffle test method**
  - It should fail whenever we have new Random(constant)
    - e.g. new Random(0), new Random(1), new Random(42)
- **Use the ideas from our testShuffleChars()**

ArrayShuffler.shuffle() Methods static

- **The shuffle() methods are utility methods**
  - ArrayShuffler itself has no state
- **It would thus be reasonable to make them static**
  - Our calling code still works as-is

```
public class ArrayShuffler {
    public static void shuffle(int[] values) { /* ... */ }
    public static void shuffle(char[] values) { /* ... */ }
}
```

Forcing Use of Static Methods

- **ArrayShuffler is a utility class**
  - It has no state
- **We should stop users from making instances**
- **Simply add a private constructor**
  - This will find all the places where we are making instances

```
public class ArrayShuffler {
    private ArrayShuffler() { }
    public static void shuffle(int[] values) {
        // ...
```

Avoid duplicate code

- **ArrayShuffler has two identical shuffle methods**
  - This is necessary because the int[] and char[] are unrelated
- **We can use java.lang.reflect.Array instead:**
  - Array.getLength(values)
  - Array.get(values, index)
  - Array.set(values, index, newValue)

ArrayShuffler.shuffleInternal()

```
public static void shuffle(int[] values) {
    shuffleInternal(values);
}
public static void shuffle(char[] values) {
    shuffleInternal(values);
}
private static void shuffleInternal(Object values) {
    Random random = new Random();
    int length = Array.getLength(values);
    for (int i = length - 1; i > 0; i--) {
        int swap = random.nextInt(i + 1);
        if (swap != i) {
            Object temp = Array.get(values, i);
            Array.set(values, i, Array.get(values, swap));
            Array.set(values, swap, temp);
        }
    }
}
```

Inner Class ActionListener

- **Anagrams has private nested classes for the actions**

```
public Anagrams() {
    // ...
    guess.addActionListener(new GuessActionListener());
    // ...
}

private class GuessActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        guess();
    }
}
```

ActionListener Anonymous Types

- The GuessActionListener can become anonymous

```
public Anagrams() {
    // ...
    guess.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            guess();
        }
    });
    // ...
}
```

Converting Anonymous Types to

- Java 8 introduced lambdas to reduce clutter

```
public Anagrams() {
    // ...
    guess.addActionListener(e -> guess());
    // ...
}
```

Font Ligatures

- Some IDEs use font ligatures to make code prettier

Normal	Ligatures
!=	≠
<=	⩽
>=	⩾
->	→

```
guess.addActionListener(e → guess());
```

Make Classes Final Where Possible

- **Better to start final and then open up if needed**
  - Once someone extends our class, we have to maintain it
- **All should be final except ScrambledWordLibrary**
  - Even the test classes should be final
- **The "blessed" order for modifiers for classes is**

```
ClassModifier:
(one of)
Annotation public protected private
abstract static final sealed non-sealed strictfp
```

Sealing the abstract class & interface

- **We state that classes are sealed (Java 17)**
  - And which classes are permitted to subclass them
  - The stricter we are, the better for our future sanity

```
public sealed interface WordLibrary
        permits ScrambledWordLibrary,
                ShuffledWordLibrary,
                StaticWordLibrary {
    // ...
}
public abstract sealed class ScrambledWordLibrary
        implements WordLibrary
            permits RandomScrambledWordLibrary,
                    SortedScrambledWordLibrary {
    // ...
}
```

Commit and Final Push

- **One last commit and then I will push my changes**
  - Remember, all my work is in the "kabutz" branch
- **Please commit your code to your branch**
- **As soon as possible, create own GitHub account**
  - Fork projects that you find interesting
  - Create your own projects
  - Start building up your portfolio of work
    - Show your best creations to the world

Conclusion

- **The fastest way to learn a language is to speak it**
  - Learning a programming language is like a natural language
  - Easiest is to immerse yourself in the language
    - Make mistakes, laugh, try again
    - "Και ένα κιλό λαχανικά σας παρακαλώ."

The Java Specialists' Newsletter

- **Make sure to subscribe**
  - www.javaspecialists.eu/archive/subscribe/
- **Readers in 150+ countries**
- **Over 21 years of newsletters on advanced Java**
  - All previous newsletters available on www.javaspecialists.eu
  - Courses, additional training, etc.