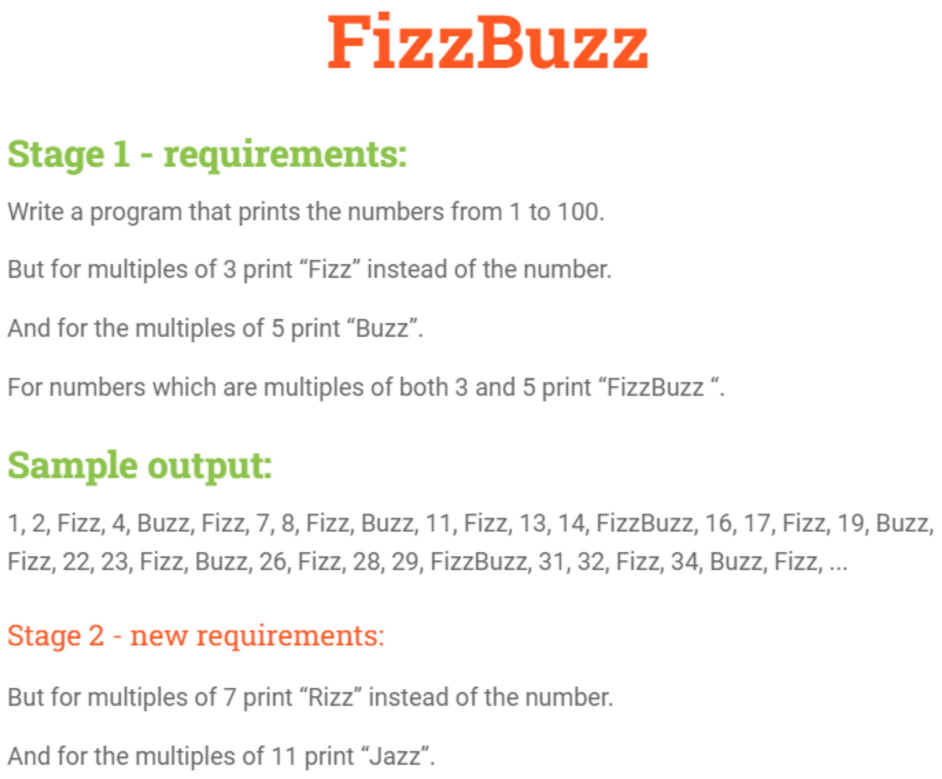
Curs 1:



FizzBuzz

Stage 1 - requirements:

Write a program that prints the numbers from 1 to 100.

But for multiples of 3 print “Fizz” instead of the number.

And for the multiples of 5 print “Buzz”.

For numbers which are multiples of both 3 and 5 print “FizzBuzz “.

Sample output:

1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, FizzBuzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, FizzBuzz, 31, 32, Fizz, 34, Buzz, Fizz, ...

Stage 2 - new requirements:

But for multiples of 7 print “Rizz” instead of the number.

And for the multiples of 11 print “Jazz”.

..

FooBarQix

Stage 1 - requirements:

You should implement a function `String compute(int)` which implements the following rules:

1. If the number is divisible by 3, write “Foo” instead of the number
2. If the number is divisible by 5, add “Bar”
3. If the number is divisible by 7, add “Qix”
4. For each digit 3, 5, 7, add “Foo”, “Bar”, “Qix” in the digits order.

Sample output:

- 1 => 1
- 2 => 2
- 3 => FooFoo (divisible by 3, contains 3)
- 4 => 4
- 5 => BarBar (divisible by 5, contains 5)
- 6 => Foo (divisible by 3)
- 7 => QixQix (divisible by 7, contains 7)
- 8 => 8
- 9 => Foo
- 10 => Bar
- 13 => Foo

…

Pair of 2

Stage 1 - requirements:

Given an array of integers, find the number of pairs from the array.

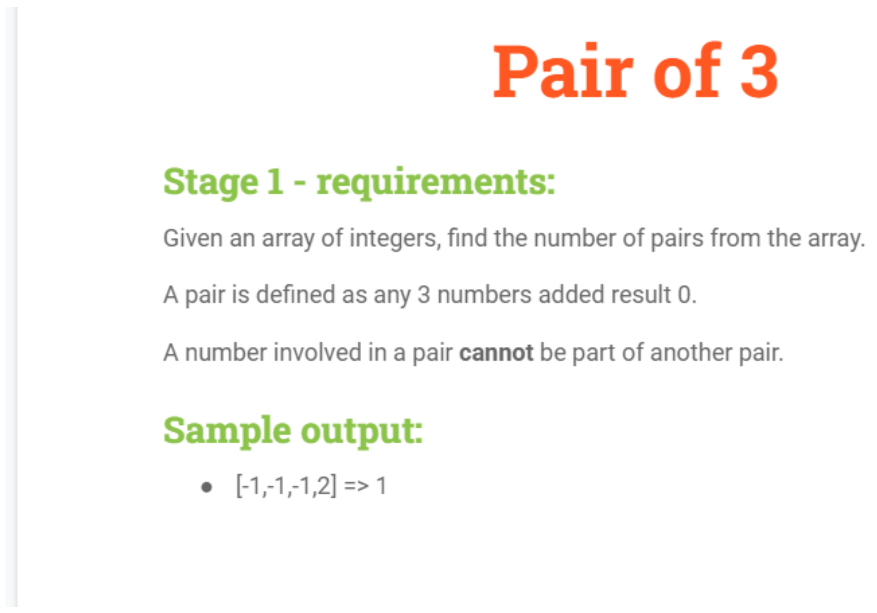
A pair is defined as any 2 numbers added result 0.

A number involved in a pair **cannot** be part of another pair.

Sample output:

- [3,2,-3,-2,3,0] => 2
- [1,1,0,-1,-1] => 2
- [5,9,-5,7,-5] => 1

..



Pair of 3

Stage 1 - requirements:

Given an array of integers, find the number of pairs from the array.

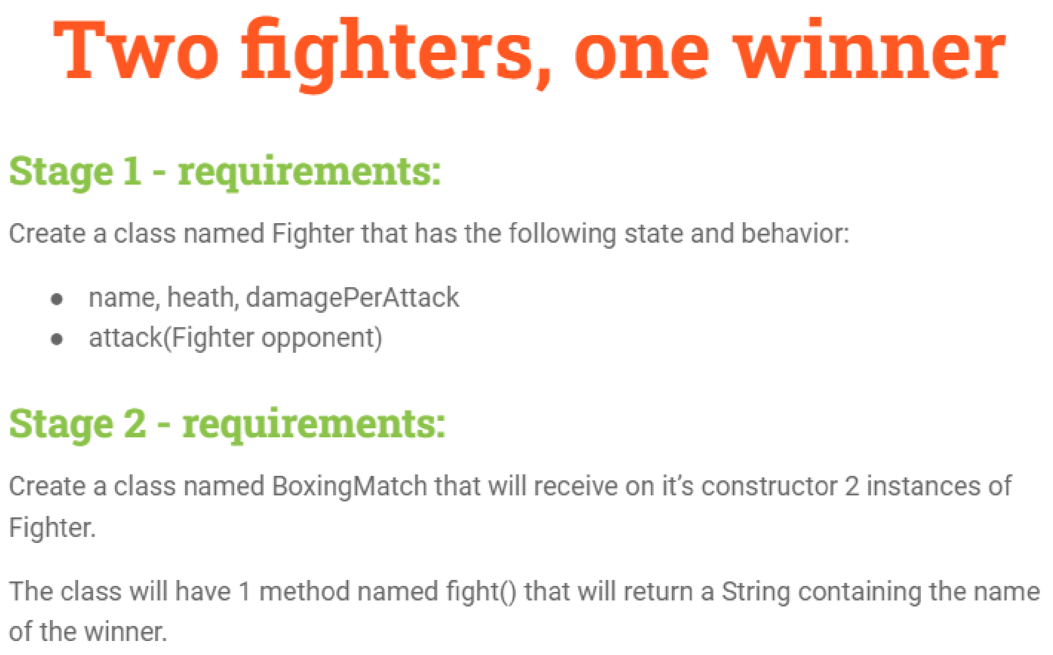
A pair is defined as any 3 numbers added result 0.

A number involved in a pair **cannot** be part of another pair.

Sample output:

- [-1,-1,-1,2] => 1

Curs 2:



Two fighters, one winner

Stage 1 - requirements:

Create a class named Fighter that has the following state and behavior:

- name, heath, damagePerAttack
- attack(Fighter opponent)

Stage 2 - requirements:

Create a class named BoxingMatch that will receive on it's constructor 2 instances of Fighter.

The class will have 1 method named fight() that will return a String containing the name of the winner.

..

Movie Database

Stage 1 - requirements:

Create the following classes:

- Premiu(nume: String, an: Integer)
- Actor(nume: String, varsta: Integer, premii: Premiu[])
- Film(anAparitie: Integer, nume: String, actori: Actor[])
- Studio(nume: String, filme: Film[])

Stage 2 - requirements:

Create your own object instances and relationships between them.

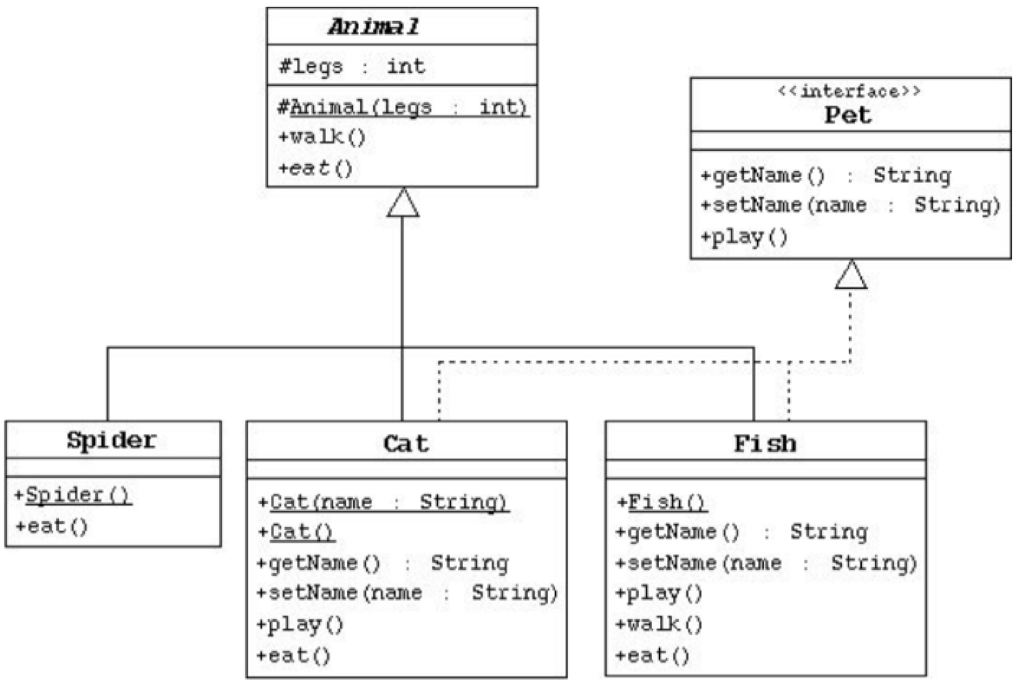
Also for this example you can use the following:

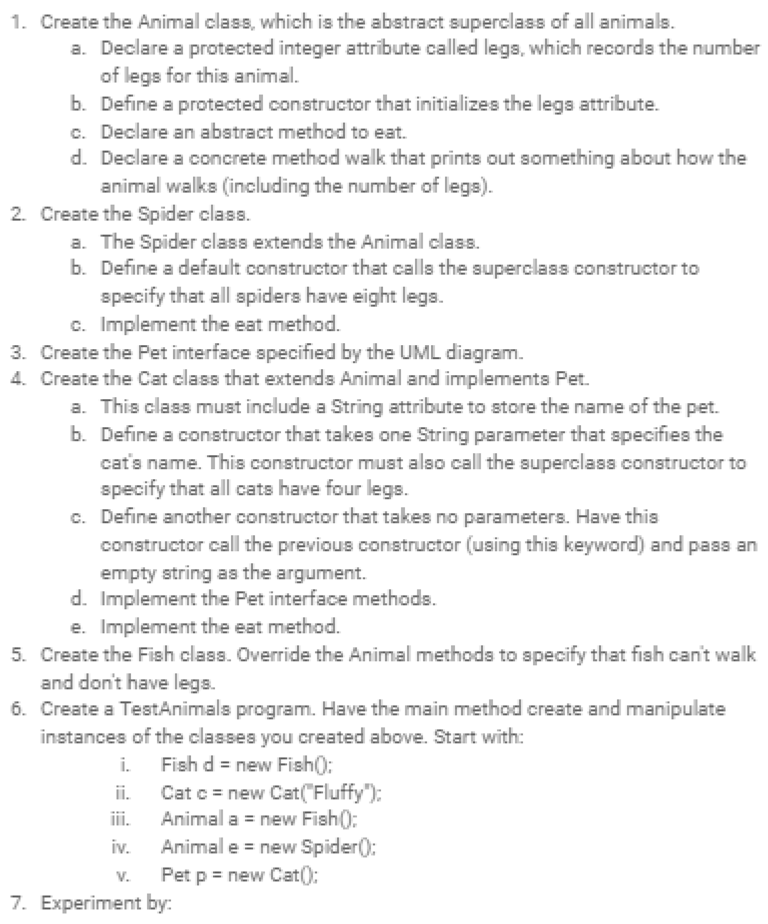
```
Premiu oscar1990 = new Premiu("oscar", 1990);
Premiu oscar2000 = new Premiu("oscar", 2000);
Premiu oscar2010 = new Premiu("oscar", 2010);
Premiu oscar2018 = new Premiu("oscar", 2018);

Actor actorOscar1990 = new Actor("actor cu 2 oscaruri", 35, new
Premiu[]{oscar1990, oscar2000});
Actor actorOscar2010 = new Actor("actor cu oscar din 2000", 55, new
Premiu[]{oscar2010});
Actor actorOscar2018 = new Actor("actor cu oscar din 2018", 23, new
Premiu[]{oscar2018});
Actor actorFaraPremii01 = new Actor("actor fara oscar 01", 33, new
Premiu[]{});
```

...

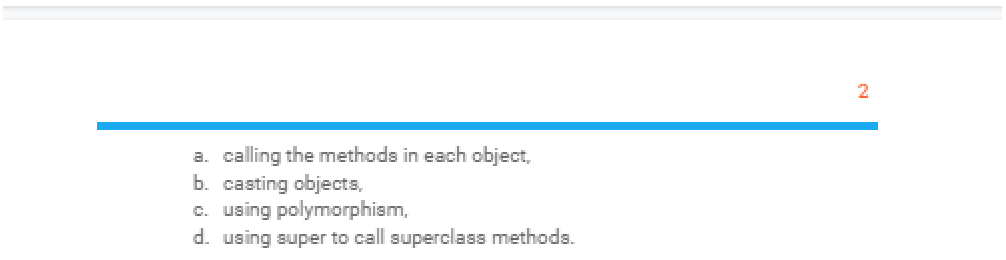
Animal Hierarchy

Stage 1 - requirements:





1. Create the Animal class, which is the abstract superclass of all animals.
   a. Declare a protected integer attribute called legs, which records the number of legs for this animal.
   b. Define a protected constructor that initializes the legs attribute.
   c. Declare an abstract method to eat.
   d. Declare a concrete method walk that prints out something about how the animal walks (including the number of legs).
2. Create the Spider class.
   a. The Spider class extends the Animal class.
   b. Define a default constructor that calls the superclass constructor to specify that all spiders have eight legs.
   c. Implement the eat method.
3. Create the Pet interface specified by the UML diagram.
4. Create the Cat class that extends Animal and implements Pet.
   a. This class must include a String attribute to store the name of the pet.
   b. Define a constructor that takes one String parameter that specifies the cat's name. This constructor must also call the superclass constructor to specify that all cats have four legs.
   c. Define another constructor that takes no parameters. Have this constructor call the previous constructor (using this keyword) and pass an empty string as the argument.
   d. Implement the Pet interface methods.
   e. Implement the eat method.
5. Create the Fish class. Override the Animal methods to specify that fish can't walk and don't have legs.
6. Create a TestAnimals program. Have the main method create and manipulate instances of the classes you created above. Start with:
   i. Fish d = new Fish();
   ii. Cat c = new Cat("Fluffy");
   iii. Animal a = new Fish();
   iv. Animal e = new Spider();
   v. Pet p = new Cat();
7. Experiment by:



   a. calling the methods in each object,
   b. casting objects,
   c. using polymorphism,
   d. using super to call superclass methods.

Curs 3:

Iterator

Stage 1 - requirements:

Implement the iterator pattern for an integer array. The Iterator class should be named ArrayCustomIterator and should have 2 methods:

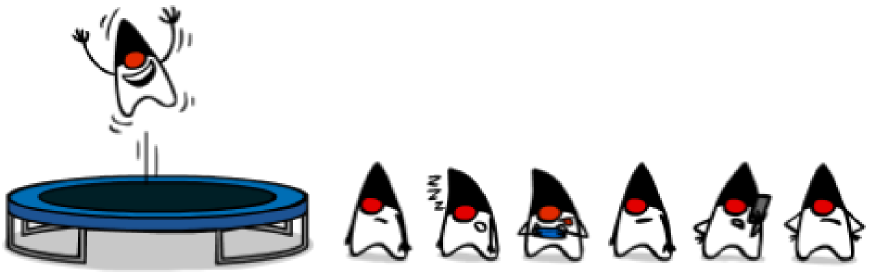
```
boolean hasNext();
int next();
```

Sample output:

The following sample code should print 1,2,3.

```
int[] arr = new int[] {1,2,3};
ArrayCustomIterator it = new ArrayCustomIterator(arr);
while (it.hasNext()) {
      System.out.println(it.next());
}
```

…



Strategy

Stage 1 - requirements:

Given an interface SortingStrategy with the following method:

```
      void sort(Integer[] list);
```

Implement 2 sorting strategies BubleSort and MergeSort then use the following displaySorted method signature to apply them:

```
      void displaySorted(SortingStrategy strategy, Integer[] arr)
```

Sample output:

The following sample code should print 1 2 3 4 5 6 7 8 9 .

```
public static void main(String[] args) {
    Integer[] arr = new Integer[]{1, 4, 5, 2, 3, 6, 9, 8, 7};
    Integer[] copy01OfArr = Arrays.copyOf(arr, arr.length);
    Integer[] copy02OfArr = Arrays.copyOf(arr, arr.length);

    displaySorted(new BubleSort(), copy01OfArr);
    displaySorted(new MergeSort(), copy02OfArr);
  }
```

…

Proxy

Stage 1 - requirements:

Given the following scenario “a student wants to rent an apartment from a real estate agent”, make use of the proxy design pattern and implement this scenario in an OO way.

The real estate agent had some issues in the past with students that have their names starting with the letter “P” and now they refuse to rent apartments to anybody who’s name starts with the letter “P”.

Implement a class Apartment(String location, int monthlyRentCost) and a class Student(String name, int money).

The RealEstateAgentProxy has 2 methods:

```
          void represent(Apartment appartment);
          Apartment rent(Student student);
```

Sample output:

The following sample code should print :

```
Student{name='Ionescu', money=500} rented
Apartment{location='Crangasi01', monthlyRentCost=200}

Student{name='Popescu', money=330} rented null
```

…

Observer

Stage 1 - requirements:

Imagine the following scenario “a teacher in front of a class full of students”. When the teacher teaches, the students receive the info that the teacher is sharing with them.

The teacher is the “observed” subject and students “observe” the teacher.

Given the interfaces below, implement Student class and Teacher class so that the following main method will print the expected message.

```
public interface ObservedSubject {
  void register(Observer obj);
  void unregister(Observer obj);
  void notifyObservers(String message);
}
public interface Observer {
  void update(String message);
}
```

…

Template method

Stage 1 - requirements:

Create 2 concrete implementations of the following abstract class: AscBubleSort and DescBubleSort. One will sort in ascending order and the oher in descending order.

```
public abstract class TemplateMethodBubleSort {

  void sort(Integer[] list) {
     //make use of method numbersInCorrectOrder to sort array
     ...
  }

  abstract boolean numbersInCorrectOrder(Integer i1, Integer i2);
}
```

…

Builder

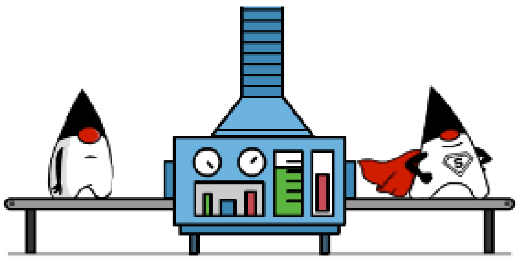
Stage 1 - requirements:

Given a Person class with the following fields: name, job, university, drivingLicese, isMaried.

When creating a new Person, only the name field is required, all the others are optional.

Create a builder class that will help you create instances of Person class.

...



Decorations

Stage 1 - requirements:

Using the decorator design pattern, decorate the following ChristmasTree with Bulbs, Candy and Garland.

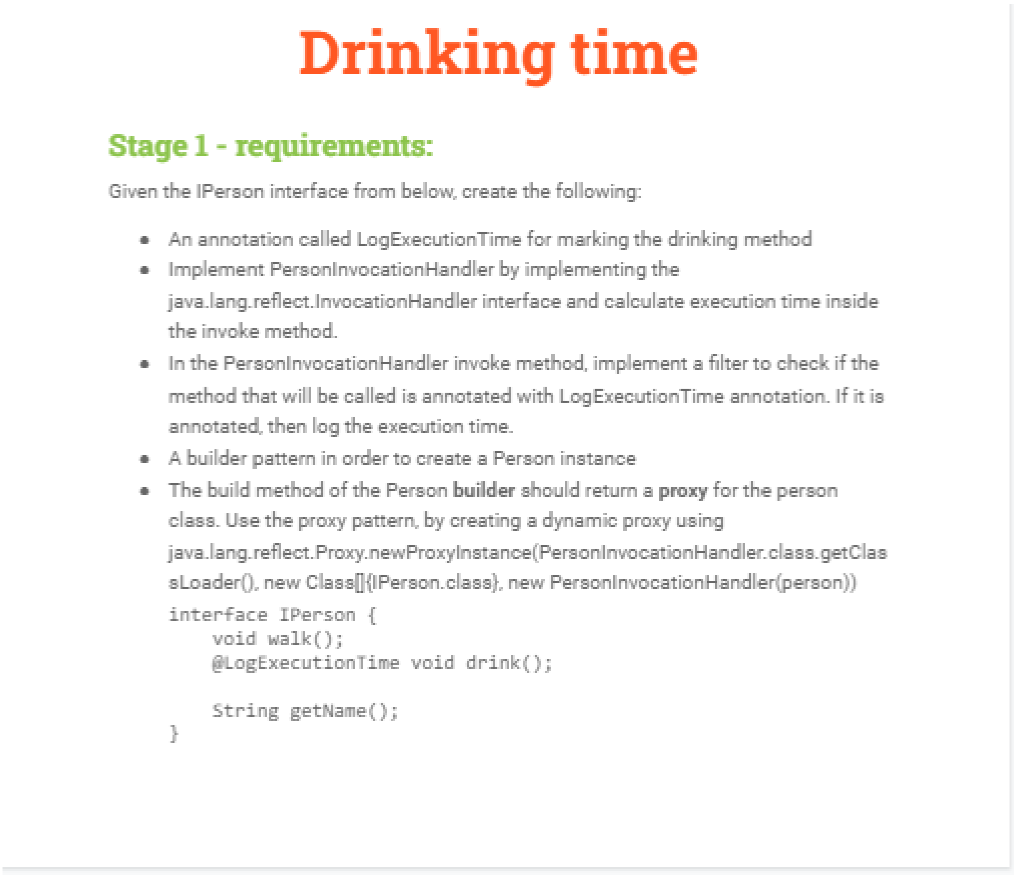
ORIGINAL TREE	DECORATED TREE
`         <>`	`         <>`
`        <  >`	`        <  >`
`       <    >`	`       <    >`
`      <      >`	`      <CCCCCC>`
`     <        >`	`     <        >`
`    <          >`	`    <          >`
`   <            >`	`   <BBBBBBBBBBBB>`
`  <              >`	`  <              >`
` <                >`	` <GGGGGGGGGGGGGGGG>`
`<                  >`	`<                  >`
`       |    |`	`       |    |`
`       |    |`	`       |    |`
`       |    |`	`       |    |`

For the next main method and described classes you should have an output like following:

```
public class DecorationsExample {
    public static void main(String[] args) {
        DecorableTree christmasTree = new ChristmasTree();
        DecorableTree decoratedTree =
                      new Garland(new Bulb(new Candy(christmasTree)));
        decoratedTree.display();
        christmasTree.display();
    }
}
```

```java
interface DecorableTree {
    List<List<String>> getTree();
    void display();
}
class ChristmasTree implements DecorableTree {
    private List<List<String>> tree;
    private int size;
    public ChristmasTree() { this.size = 10; this.tree = getTree(); }
    public List<List<String>> getTree() {
        List<List<String>> tree = new ArrayList<>();
        List<String> row;
        for (int i = 0; i < size; i++) {
            row = new ArrayList<>();
            for (int j = 0; j < size * 2; j++) {
                if (j == (size - i)) row.add("<");
                if (j == (size + i)) row.add(">");
                row.add(" ");
            }
            tree.add(Collections.unmodifiableList(row));
        }
        for (int i = 0; i < 3; i++) {
            row = new ArrayList<>();
            for (int j = 0; j < size * 2; j++) {
                if (j == size - 2) row.add("|");
                if (j == size + 2) row.add("|");
                row.add(" ");
            }
            tree.add(Collections.unmodifiableList(row));
        }
        return Collections.unmodifiableList(tree);
    }
    public void display() {
        for (int i = 0; i < tree.size(); i++) {
            for (int j = 0; j < tree.get(i).size(); j++) {
                System.out.print(tree.get(i).get(j));
            }
            System.out.println();
        }
    }
}
```

…



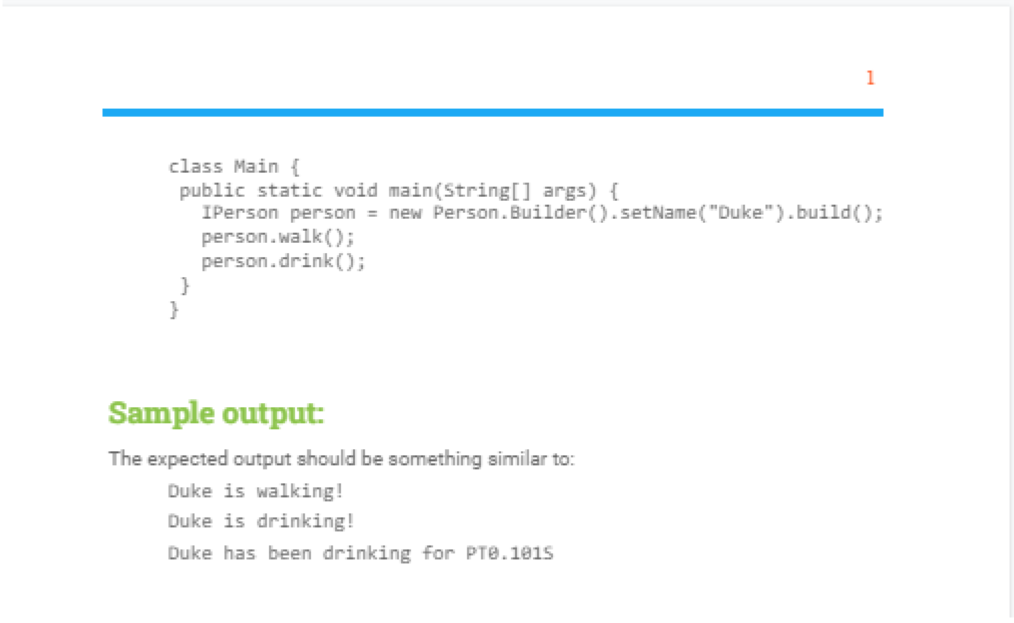
Drinking time

Stage 1 - requirements:

Given the IPerson interface from below, create the following:

- An annotation called LogExecutionTime for marking the drinking method
- Implement PersonInvocationHandler by implementing the java.lang.reflect.InvocationHandler interface and calculate execution time inside the invoke method.
- In the PersonInvocationHandler invoke method, implement a filter to check if the method that will be called is annotated with LogExecutionTime annotation. If it is annotated, then log the execution time.
- A builder pattern in order to create a Person instance
- The build method of the Person **builder** should return a **proxy** for the person class. Use the proxy pattern, by creating a dynamic proxy using java.lang.reflect.Proxy.newProxyInstance(PersonInvocationHandler.class.getClassLoader(), new Class[]{IPerson.class}, new PersonInvocationHandler(person))

```
interface IPerson {
    void walk();
    @LogExecutionTime void drink();

    String getName();
}
```



1

```
class Main {
 public static void main(String[] args) {
   IPerson person = new Person.Builder().setName("Duke").build();
   person.walk();
   person.drink();
 }
}
```

Sample output:

The expected output should be something similar to:

```
Duke is walking!
Duke is drinking!
Duke has been drinking for PT0.101S
```

Curs 4:

Phonebook

Stage 1 - requirements:

Give the following phonebook as a text file:
https://raw.githubusercontent.com/miualinionut/java-training/master/_4_exceptions_io/_test_files/in/phonebook.txt

Find the phone number of Abbey and Abdul.

Sample output:

Input:

Abbey 8402440374

Abbie 6605142833

Abby 1071474049

Abdul 8069932296

Output: 8402440374, 8069932296

https://raw.githubusercontent.com/miualinionut/java-training/master/_4_exceptions_io/_test_files/in/phonebook.txt

….

Secret message

Stage 1 - requirements:

Discover the secret message from the following text by following these instructions:

- Take into consideration only capital letters from the input text
- Consider capital X as space between words

Input text file:
https://raw.githubusercontent.com/miualinionut/java-training/master/_4_exceptions_io/_test_files/in/message.txt

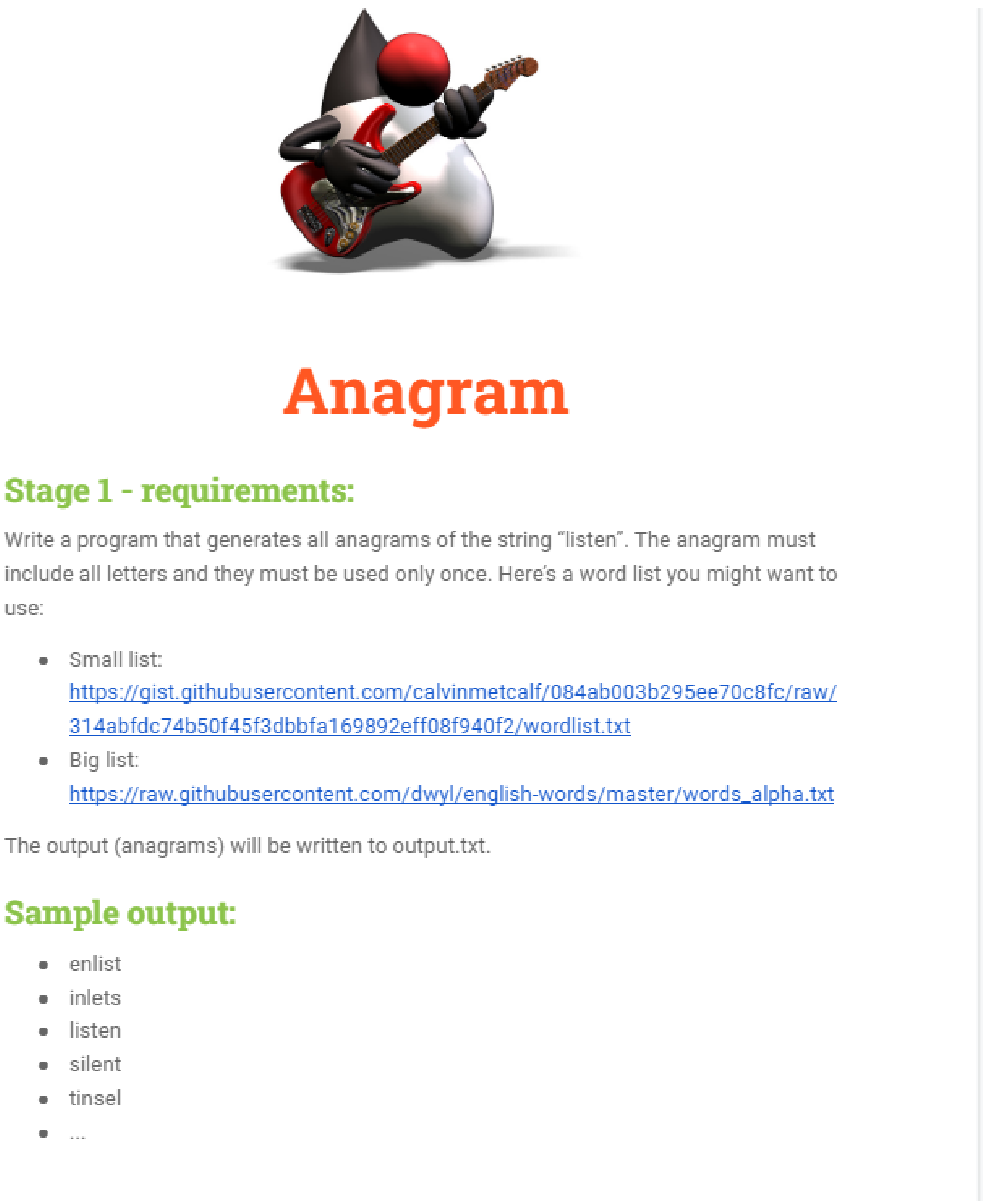
Sample output:

Input: "t**H**e quick brown fox jumps over the lazy dog. th**E** quick brown fox jumps over the **L**azy dog. the quick brown fox jum**P**s over the lazy dog. the quick brown fo**X** ju**M**ps over the lazy dog. th**E** quick brown fox jumps over the lazy dog"

Output: HELP ME

https://raw.githubusercontent.com/miualinionut/java-training/master/_4_exceptions_io/_test_files/in/message.txt

…



Anagram

Stage 1 - requirements:

Write a program that generates all anagrams of the string "listen". The anagram must include all letters and they must be used only once. Here's a word list you might want to use:

- Small list: https://gist.githubusercontent.com/calvinmetcalf/084ab003b295ee70c8fc/raw/314abfdc74b50f45f3dbbfa169892eff08f940f2/wordlist.txt
- Big list: https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt

The output (anagrams) will be written to output.txt.

Sample output:

- enlist
- inlets
- listen
- silent
- tinsel
- ...

Small :
https://gist.githubusercontent.com/calvinmetcalf/084ab003b295ee70c8fc/raw/314abfdc74b50f45f3dbbfa169892eff08f940f2/wordlist.txt

Big:

https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt

...

Number to LCD

Stage 1 - requirements:

Write a program that given a number (with arbitrary number of digits), converts it into LCD style numbers. Each digit is 3 lines high.

The input will be read from an input.txt file and output will be written to output.txt.

Sample output:

```
    _  _     _  _  _  _  _ 
  | _| _||_||_ |_   ||_||_|
  ||_  _|  | _||_|  ||_| _|
```

Stage 2 - new requirements:

Change your program to support variable width or height of the digits. For example for width = 3 and height = 5 the digit 2 will be:

Sample output:

```
 ___
    |
 ___|
|
|___
```

...

Occurrence

Stage 1 - requirements:

Using Scanner, count how many times a word appears in a large text file provided as input.

Display top 10 word counts to the console after counting all.

Sample output:

For the following input:

```
      How are you today ?
      Do you think is going to rain ?
```

We will get:

```
you - 2
how - 1
...
```

Curs 5 :

Exchange Desk

Stage 1 - requirements:

Write a ExchangeDesk class to change currencies, like the foreign currency desk of an airport. Avoid hard-coding any particular currency class in the converter. Preserve currency types in the conversion.

So your goal is to change the type declarations in convert from specific currencies to general Currency references using generics.

Hard-code the exchange rate for this part.

Sample output:

The currency conversion call should look like this:

```
RON lei = new RON(1_000);
```

```
USD dollar = exchangeDesk.convert(lei, USD.class);
```

Stage 2 - new requirements:

Offer the possibility to add exchange rates.

Adding a rate for USD to RON should also add the corresponding rate for RON to USD.

Sample output:

```
exchangeDesk.addRate(RON.class, USD.class, 4.1d);
```

...

Pairs

Stage 1 - requirements:

Given the `Pair` class from below and associated classes, make the necessary changes so that the compiler will require that a pair will be formed only from the same type of shoes.

```
class Pair {
    private Object first;
    private Object second;
    public Pair(Object firstElement, Object secondElement) {
        first = firstElement;
        second = secondElement;
    }
    public Object getFirst() { return first; }
    public Object getSecond() { return second; }
}

interface Shoe {}
class Running implements Shoe {}
class Heels implements Shoe {}
class Boot implements Shoe {}
```

Sample output:

After all the necessary changes, the following code should compile successfully:

```
Running runningShoe1 = new Running("RED", 41);
```

```
    Running runningShoe2 = new Running("RED", 41);
    Pair<Running> pairOK = new Pair<>(runningShoe1, runningShoe2);
```

And the following should throw a compilation error:

```
    Running runningShoe1 = new Running("RED", 41);
    Boot bootShoe = new Boot("BLACK", 45);
    Pair<Running> pairKO = new Pair<>(runningShoe1, bootShoe);
```

Stage 2 - new requirements:

Make the necessary code changes so that the color and size of the shoe will be checked at runtime if they match. In case the colors or sizes do not match throw an exception.

*involves creation of many many classes

Sample output:

```
    Running runningShoe1 = new Running("RED", 41);
    Running runningShoe2 = new Running("RED", 42);
    Pair<Running> pairOK = new Pair<>(runningShoe1, runningShoe2);
```

⇒ will throw SizesDoNotMatchException because sizes do not match

⇒ or will throw ColorsDoNotMatchException if colors do not match

....

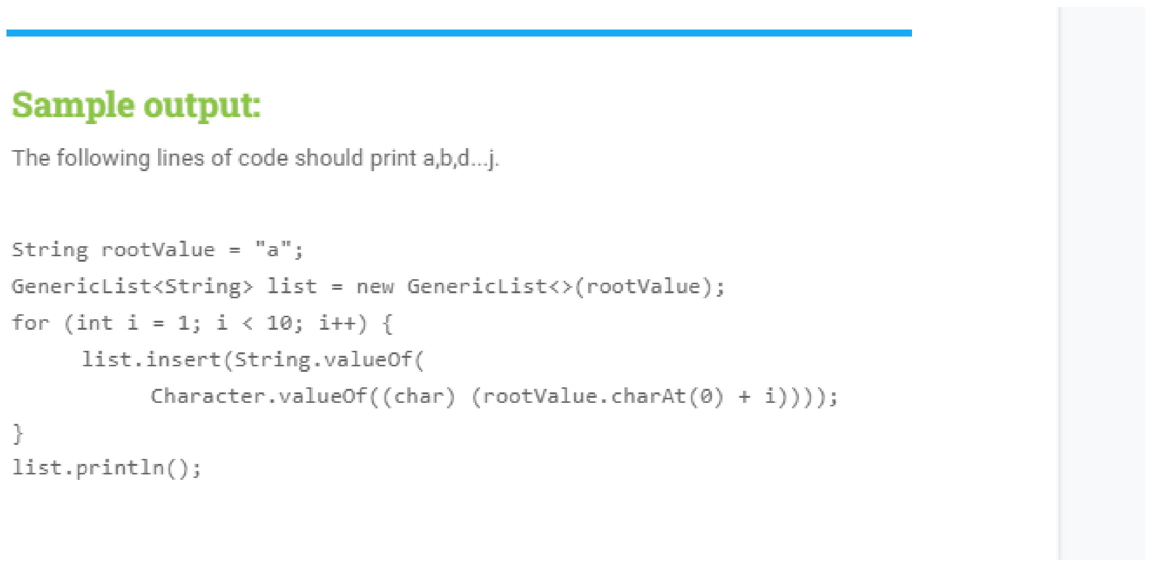
Generic Linked List

Stage 1 - requirements:

Using the following 2 interfaces create the corresponding generic implementations so that the GenericList will be able to store:

- collections formed from Integer objects
- or collections from String objects
- or any other type of objects

```
interface IGenericList<T> {
    void insert(T element);
    void println();
}
interface IGenericNode<T> {
    T getValue();
    void setValue(T value);
    IGenericNode<T> getNext();
    void setNext(IGenericNode<T> next);
}
```



Sample output:

The following lines of code should print a,b,d...j.

```
String rootValue = "a";
GenericList<String> list = new GenericList<>(rootValue);
for (int i = 1; i < 10; i++) {
      list.insert(String.valueOf(
            Character.valueOf((char) (rootValue.charAt(0) + i))));
}
list.println();
```

…

Generic Iterator

Stage 1 - requirements:

Implement an iterator class called ArrayIterator using only the interface from below and generics.

```
interface IArrayIterator<T> {
      boolean hasNext();
      T next();
}
```

Sample output:

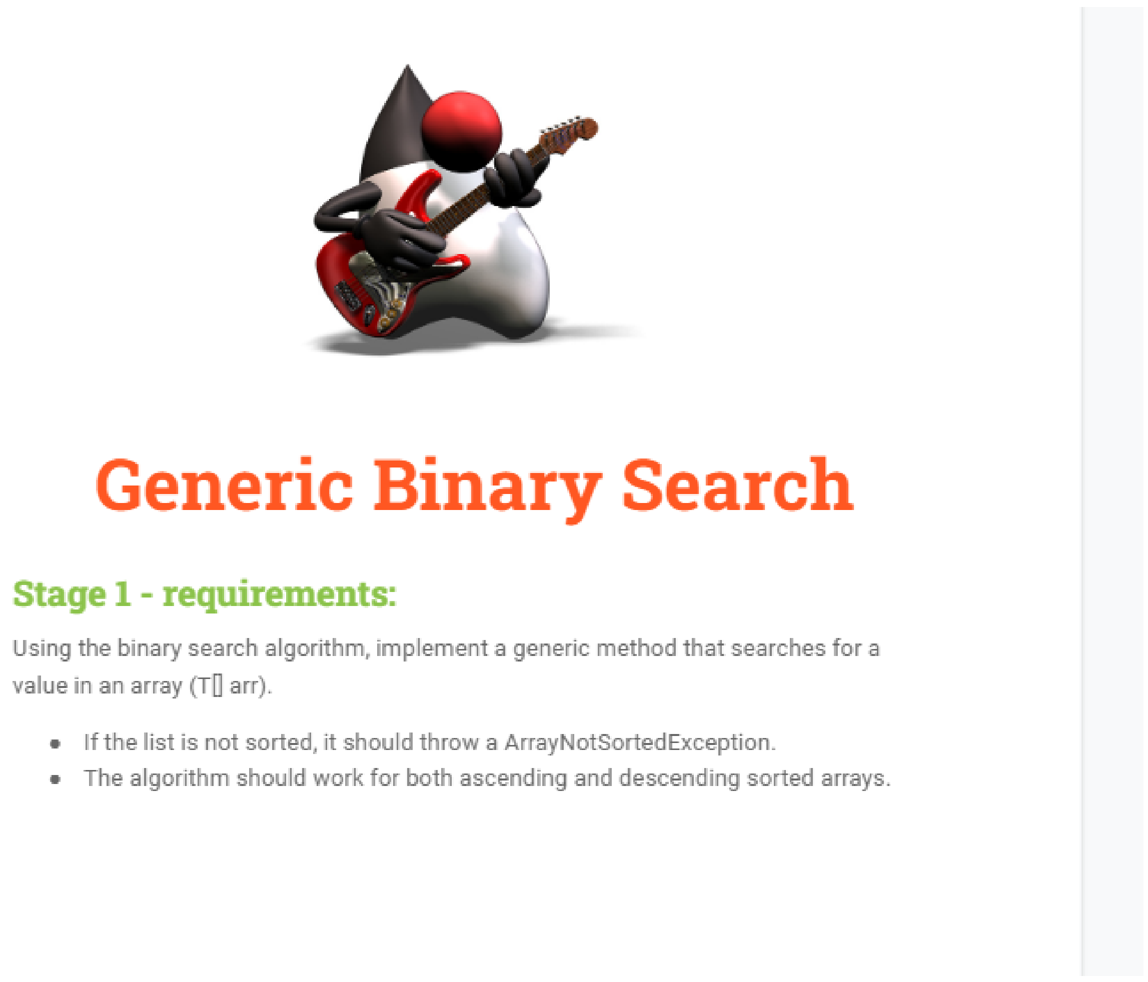
The following sample code should print 1,2,3.

```
Integer[] arr = new Integer[] {1,2,3};
IArrayIterator<Integer> it = new ArrayIterator<>(arr);
while (it.hasNext()) {
      System.out.println(it.next());
}
```

Stage 2 - new requirements:

Make the necessary code changes so that the ArrayIterator class can also work with the GenericList from the previous code challenge.

...



Generic Binary Search

Stage 1 - requirements:

Using the binary search algorithm, implement a generic method that searches for a value in an array (T[] arr).

- If the list is not sorted, it should throw a ArrayNotSortedException.
- The algorithm should work for both ascending and descending sorted arrays.

Curs 6:

Leaders

Stage 1 - requirements:

Given a List of integers, identify all leaders from it.

An integer is a leader if it's value it's greater than all numbers from it's right.

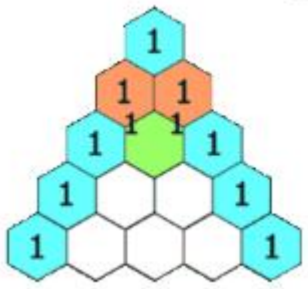
Ex: last element is always a leader.

...

Pascal's Triangle

Stage 1 - requirements:

Write a simple code that generates Pascal's triangle:



…

Zig-Zag

Stage 1 - requirements:

Given an List of integers, rearrange the integears following this rule: a < b > c < d > e < f

Sample output:

```
Input:  {4, 3, 7, 8, 6, 2, 1}
Output: {3, 7, 4, 8, 2, 6, 1}


Input:  {1, 4, 3, 2}
Output: {1, 4, 2, 3}
```

...

Merge 2 sorted Lists

Stage 1 - requirements:

Merge 2 sorted Lists.

Sample output:

```
Ex1: merge([1,5,6], [2,3,4]) => [1,2,3,4,5,6]
Ex2: merge([1,5,6,7,8,9], [2,3,4]) => [1,2,3,4,5,6,7,8,9]
```

...

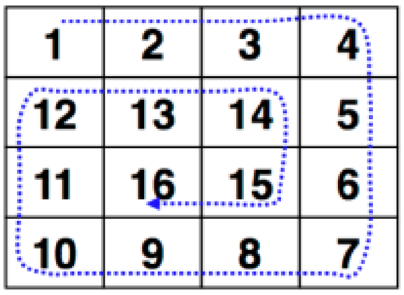
Matrix spiral

Stage 1 - requirements:

Given a square matrix, iterate in spiral and display at console the elements.

Sample output:

For the following give matrix, the algorithm will print: 1,2,3...15,16.



1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

...

MyHashTable

Stage 1 - requirements:

Write a hash table in which both the keys and the values are of type String. (do not try to write a generic class.)

Write an implementation of hash tables from scratch.

Make use of the following interface:

```
     public interface MyHashTable<K, V> {
       V get(K key);
       void put(K key, V value);
       void remove(K key);
       boolean containsKey(K key);
       int size();
     }
```

Remember that every object, has a method obj.hashCode() that can be used for computing a hash code for the object.

Sample output:

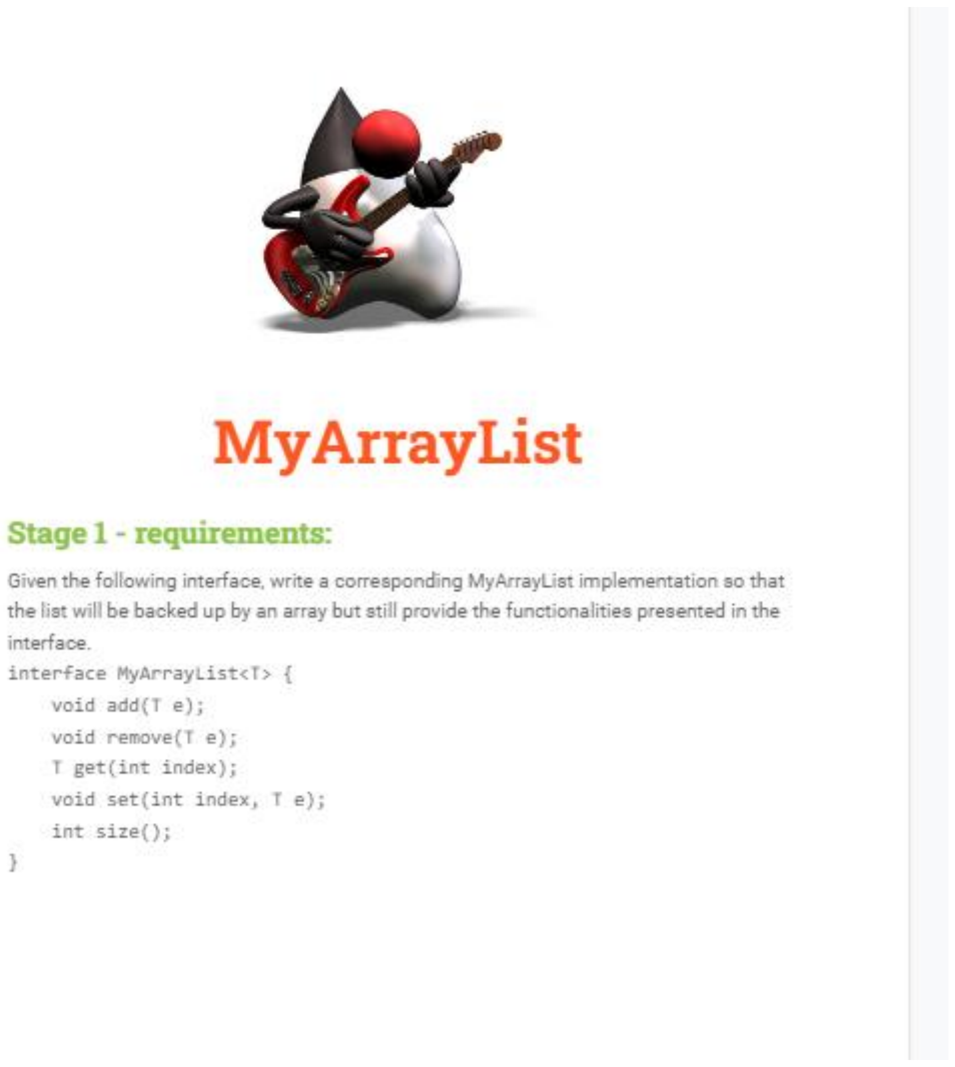
```
MyHashTable myHashTable = new MyHashTableImpl();
myHashTable.put("key1", "value1");
```

```
myHashTable.put("key2", "value2");
myHashTable.put("key3", "value3");
System.out.println(myHashTable.size()); //should print 3
```

Stage 2 - new requirements:

Modify the MyHashTable class using generics so that it will work with any type of objects, not just Strings.

....



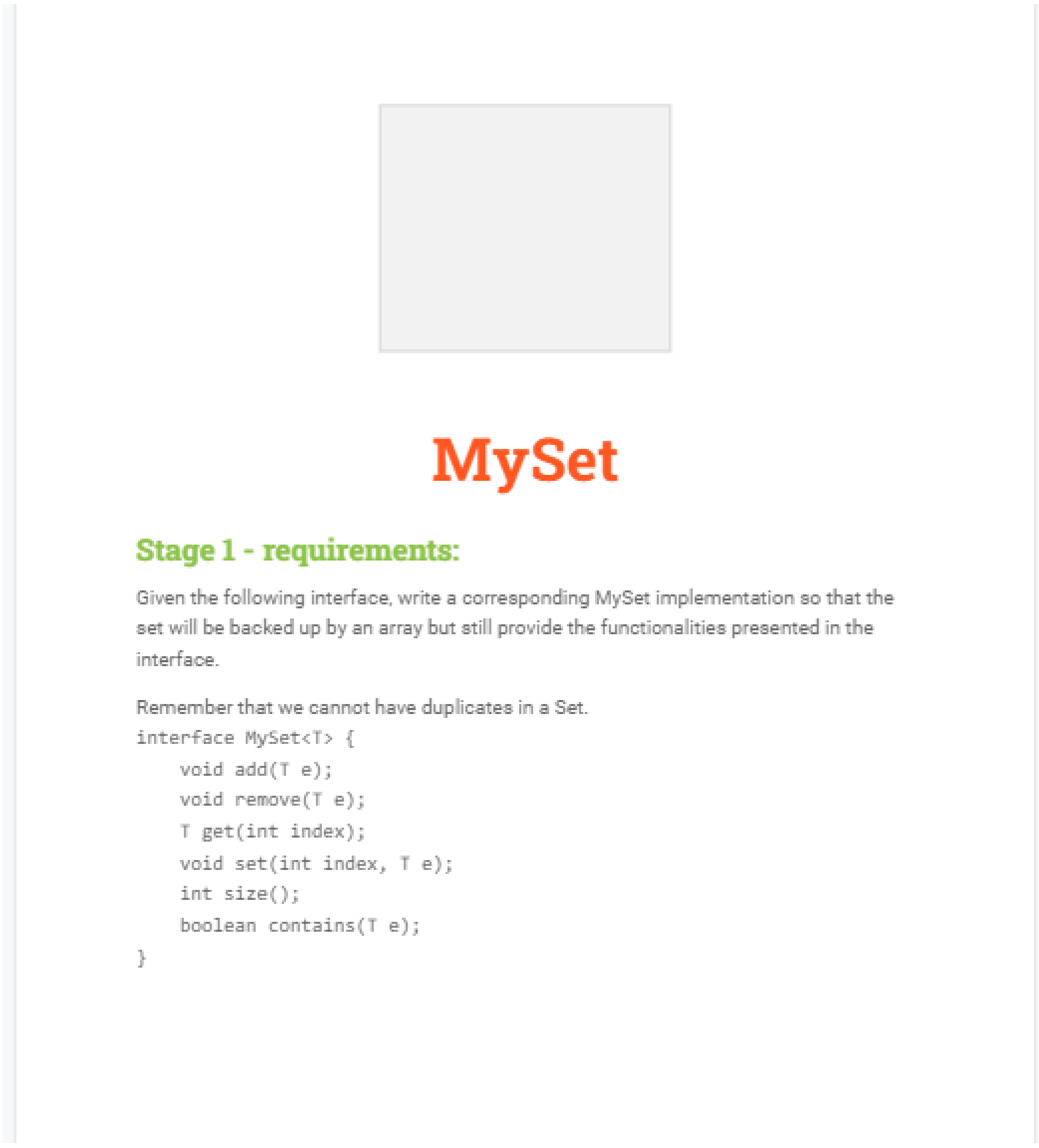
MyArrayList

Stage 1 - requirements:

Given the following interface, write a corresponding MyArrayList implementation so that the list will be backed up by an array but still provide the functionalities presented in the interface.

```
interface MyArrayList<T> {
    void add(T e);
    void remove(T e);
    T get(int index);
    void set(int index, T e);
    int size();
}
```

....



MySet

Stage 1 - requirements:

Given the following interface, write a corresponding MySet implementation so that the set will be backed up by an array but still provide the functionalities presented in the interface.

Remember that we cannot have duplicates in a Set.

```
interface MySet<T> {
    void add(T e);
    void remove(T e);
    T get(int index);
    void set(int index, T e);
    int size();
    boolean contains(T e);
}
```

Curs 7:

Rabbit Race

Stage 1 - requirements:

Implement your own RabbitThread by:

1. extending Thread class
2. constructor should receive a `int nr` as parameter so that we will be able to count the running rabbits
3. run() method should display at the console the following message: "Rabbit #nr is running"

Implement your own RabbitRunnable by:

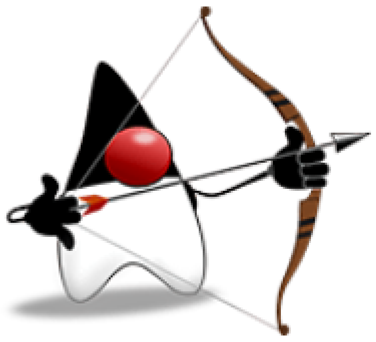
1. implementing the Runnable interface
2. constructor should receive a `int nr` as parameter so that we will be able to count the running rabbits
3. run() method should display at the console the following message: "Rabbit #nr is running"

We will put 10 rabbits in this rabbit race: 5 RabbitThreads and 5 RabbitRunnable

In order to get everything in place for the race we need to:

1. be able to create new RabbitThreads
2. be able to create new RabbitRunnable
3. align at start the rabbits in the following order: on even positions RabbitThreads and on odd positions RabbitRunnables
4. we need to be able to make the rabbits run

...



Bank Transactions

Stage 1 - requirements:

Implement your own BankAccount and TransactoinThread classes so that we can do Transactions from one account to annother:

Use the following code snippet and start making changes:

```
public class BankAccount {
  public BankAccount(String name, int debit) {
    //TODO your implementation here
  }

  void withdraw(double amount) {
    longDatabaseCall();
    //TODO your implementation here
  }

  void deposit(double amount) {
    longDatabaseCall();
    //TODO your implementation here
  }

  void longDatabaseCall() {
    try {
      Thread.sleep(100);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}

public class TransactionThread extends Thread {
  public TransactionThread(String name, BankAccount from, BankAccount
to, int amount) {
    //TODO your implementation here
  }

  @Override
  public void run() {
    //TODO your implementation here
  }

  private void transfer(BankAccount from, BankAccount to, int amount)
{
    //TODO your implementation here
  }
}
```

...

Rock Paper Scissors

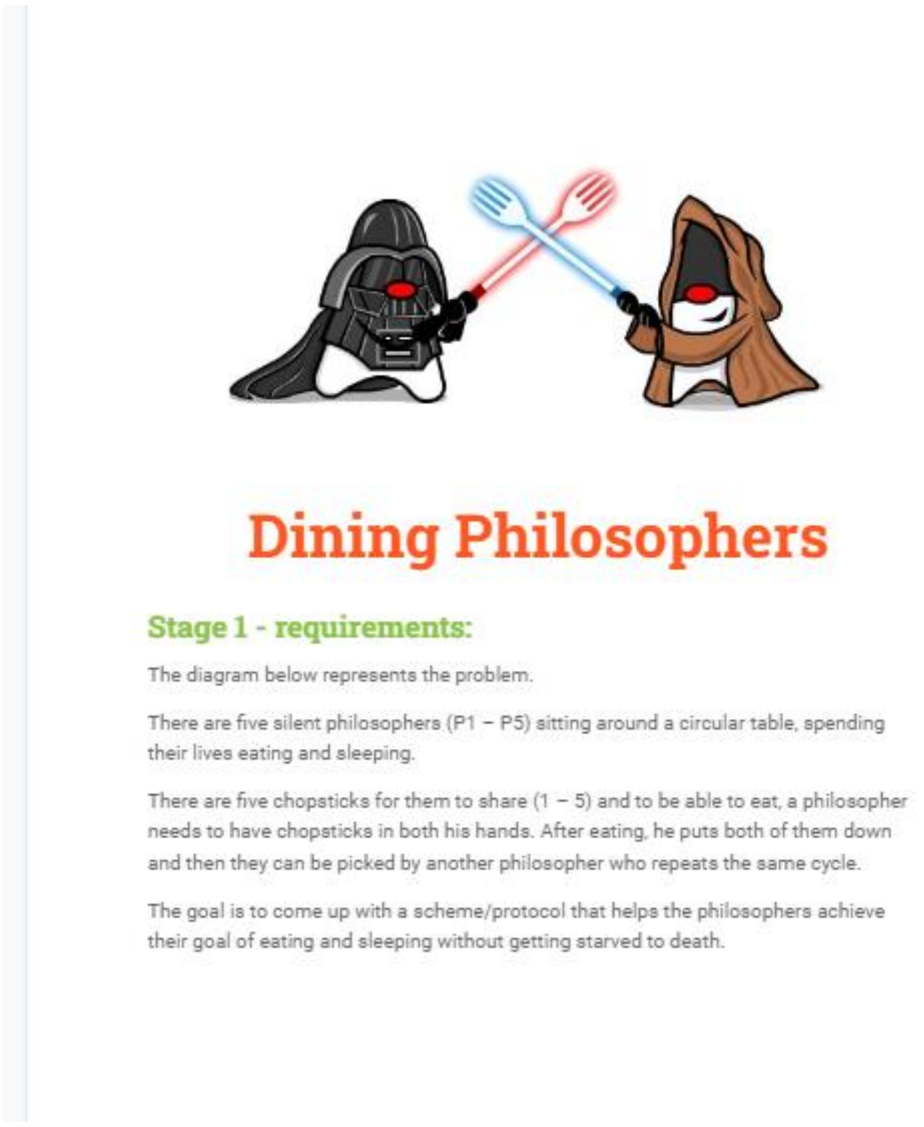
Stage 1 - requirements:

Create a class Player that extends Thread and can randomly return rock, paper or scissors.

After creating of the Player class, implement the following:

1. Spin off 2 player threads
2. Wait for them to finish choosing
3. Then display the winner.

...



Dining Philosophers

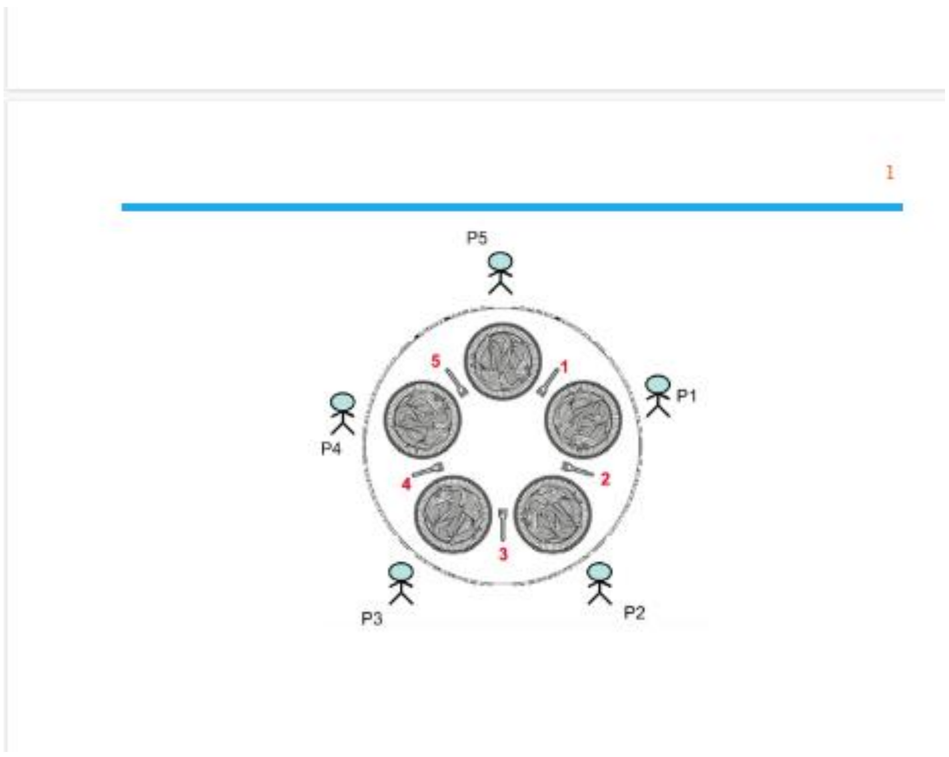
Stage 1 - requirements:

The diagram below represents the problem.

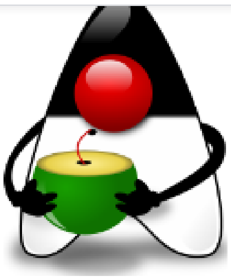
There are five silent philosophers (P1 – P5) sitting around a circular table, spending their lives eating and sleeping.

There are five chopsticks for them to share (1 – 5) and to be able to eat, a philosopher needs to have chopsticks in both his hands. After eating, he puts both of them down and then they can be picked by another philosopher who repeats the same cycle.

The goal is to come up with a scheme/protocol that helps the philosophers achieve their goal of eating and sleeping without getting starved to death.



Curs 8:



Consumer

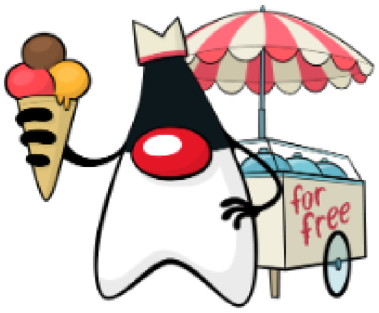
Stage 1 - requirements:

Replace the `//TODO` with the correct implementation so that the following unit tests will pass:

```
/**
* Write a consumer that clears the list it consumes.
*/
@Test
public void consumer_1() {
   List<String> list = new ArrayList<>(Arrays.asList("a", "b", "c"));

   Consumer<List<String>> consumer = null; //TODO
   consumer.accept(list);

   Assert.assertEquals(list, new ArrayList<>());
}

/**
* Write a consumer that first consumes the list with the
* consumer c1, and then consumers it with the second consumer
* c2.
*/
@Test
public void consumer_2() {
   List<String> list1 = new ArrayList<>(Arrays.asList("a", "b", "c"));
   List<String> list2 = new ArrayList<>(Arrays.asList("a", "b", "c", "first",
"second"));

   Consumer<List<String>> c1 = list -> list.add("first");
```

```
   Consumer<List<String>> c2 = list -> list.add("second");

   Consumer<List<String>> consumer = null; //TODO
   consumer.accept(list1);

   Assert.assertEquals(list1, list2);
}
```

…



Producer

Stage 1 - requirements:

Replace the `//TODO` with the correct implementation so that the following unit tests will pass:

```
/**
 * Given the predicate p1, write a predicate that
 * returns true if the string it tests is not empty.
 * This is a NOT operation on the predicate p1.
 */
@Test
public void predicate_1() {
    Predicate<String> p1 = s -> s.isEmpty();
    Predicate<String> notPredicate = null; //TODO
    Assert.assertFalse(notPredicate.test(""));
    Assert.assertTrue(notPredicate.test("Not empty!"));
}

/**
 * Given the two predicates p1 and p2, write a predicate
 * that returns true is the string it tests is
 * neither null, neither empty.
 * This is a AND operation on the predicates p1 and p2.
 */
@Test
public void predicate_2() {
    Predicate<String> p1 = s -> s != null;
    Predicate<String> p2 = s -> !s.isEmpty();
```

```
        Predicate<String> p3 = null; //TODO

        Assert.assertFalse(p3.test(""));
        Assert.assertFalse(p3.test(null));
        Assert.assertTrue(p3.test("Not empty!"));
    }

    /**
     * Given the two predicates p1 and p2, write a predicate that
     * returns true if the tested string is of length 4, true if
     * it starts with a J, but false if it is of length 4 and starts
     * with a J. This is a XOR operation on the predicates p1 and p2.
     */
    @Test
    public void predicate_3() {
        Predicate<String> p1 = s -> s.length() == 4;
        Predicate<String> p2 = s -> s.startsWith("J");

        Predicate<String> p3 = null; //TODO

        Assert.assertTrue(p3.test("True"));
        Assert.assertTrue(p3.test("Julia"));
        Assert.assertFalse(p3.test("Java"));
    }
```

....

Lambda

Stage 1 - requirements:

Exercise 1: Create a string that consists of the first letter of each word in the list of Strings provided.

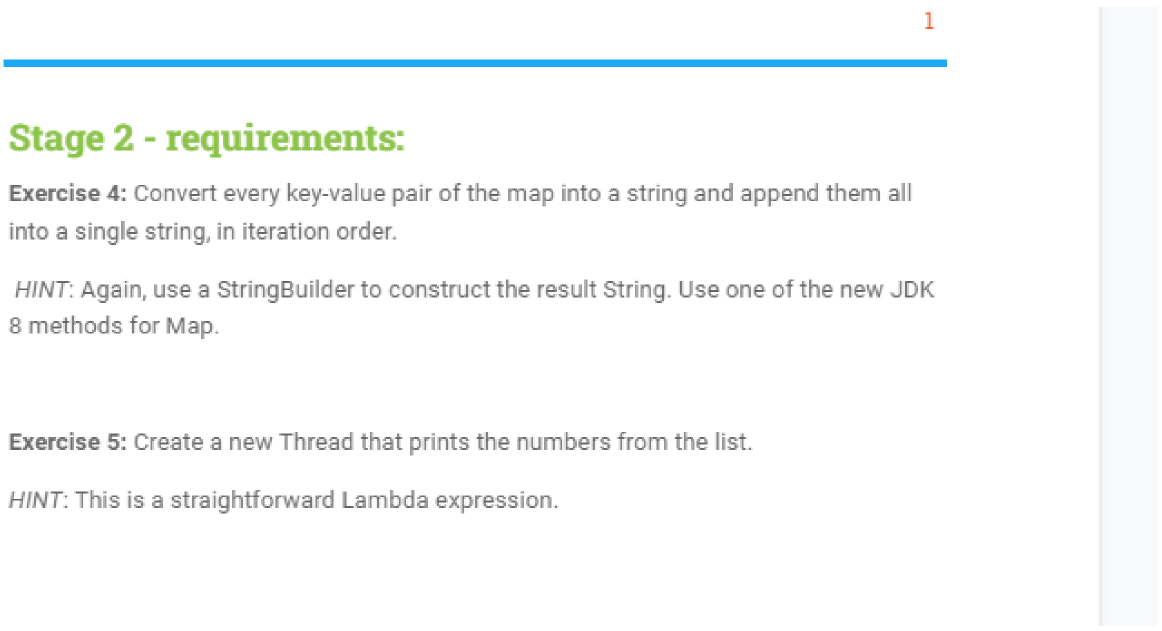
HINT: Use a StringBuilder to construct the result.

Exercise 2: Remove the words that have odd length s from the list.

HINT: Use one of the new methods from JDK 8.

Exercise 3: Replace every word in the list with its upper case equivalent.

HINT: Again, use one of the new methods from JDK 8.


1

Stage 2 - requirements:

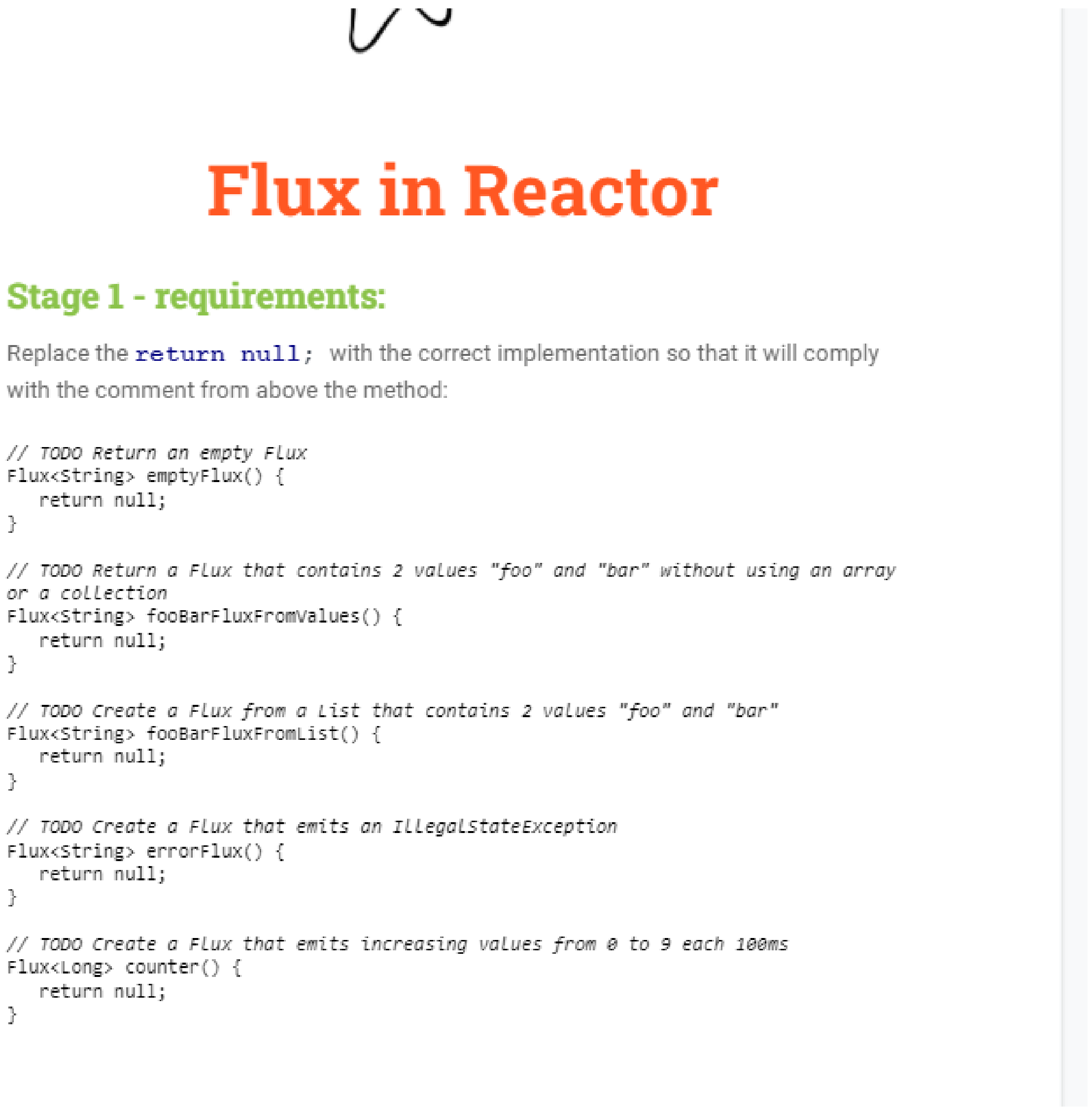
Exercise 4: Convert every key-value pair of the map into a string and append them all into a single string, in iteration order.

HINT: Again, use a StringBuilder to construct the result String. Use one of the new JDK 8 methods for Map.

Exercise 5: Create a new Thread that prints the numbers from the list.

HINT: This is a straightforward Lambda expression.

Curs 9 :



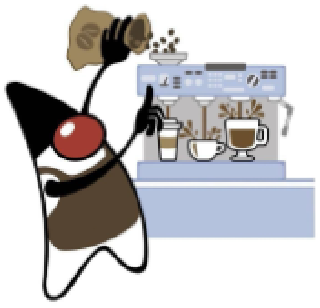
Flux in Reactor

Stage 1 - requirements:

Replace the `return null;` with the correct implementation so that it will comply with the comment from above the method:

```
// TODO Return an empty Flux
Flux<String> emptyFlux() {
   return null;
}

// TODO Return a Flux that contains 2 values "foo" and "bar" without using an array
or a collection
Flux<String> fooBarFluxFromValues() {
   return null;
}

// TODO Create a Flux from a List that contains 2 values "foo" and "bar"
Flux<String> fooBarFluxFromList() {
   return null;
}

// TODO Create a Flux that emits an IllegalStateException
Flux<String> errorFlux() {
   return null;
}

// TODO Create a Flux that emits increasing values from 0 to 9 each 100ms
Flux<Long> counter() {
   return null;
}
```

...



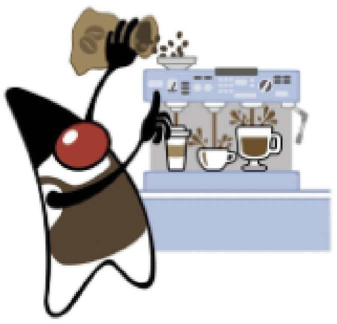
Mono in Reactor

Stage 1 - requirements:

Replace the `return null;` with the correct implementation so that it will comply with the comment from above the method:

```
// TODO Return an empty Mono
Mono<String> emptyMono() {
   return null;
}

// TODO Return a Mono that never emits any signal
Mono<String> monoWithNoSignal() {
   return null;
}

// TODO Return a Mono that contains a "foo" value
Mono<String> fooMono() {
   return null;
}

// TODO Create a Mono that emits an IllegalStateException
Mono<String> errorMono() {
   return null;
}
```

..



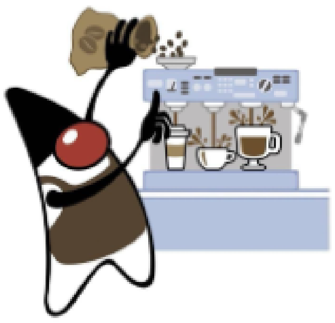
Operations in Reactor

Stage 1 - requirements:

Provide the correct implementation so that methods will comply with the comment from above them:

```
private static List<String> words = Arrays.asList(
        "the",
        "quick",
        "brown",
        "fox",
        "jumped",
        "over",
        "the",
        "lazy",
        "dog");

// TODO display the letters from the words list and count them
// 1. t
// 2. h
// 3. e
static void displayingLetters() {
}

// TODO find the missing letter from the words list
// use the same way like on displayingLetters to display the result
static void findingMissingLetter() {
}

// TODO restore the missing letter from the words list
// use the same way like on displayingLetters to display the result
static void restoringMissingLetter() {
}
```

...



Transforming in Reactor

Stage 1 - requirements:

Replace the `return null;` with the correct implementation so that it will comply with the comment from above the method:

```
// TODO Capitalize the user username, firstname and lastname
static Mono<User> capitalizeOne(Mono<User> mono) {
   return null;
}

// TODO Capitalize the users username, firstName and lastName
static Flux<User> capitalizeMany(Flux<User> flux) {
   return null;
}

// TODO Capitalize the users username, firstName and lastName using
asyncCapitalizeUser
static Flux<User> asyncCapitalizeMany(Flux<User> flux) {
   return null;
}

static Mono<User> asyncCapitalizeUser(User u) {
   return Mono.just(new User(u.getFirstName().toUpperCase(),
u.getLastName().toUpperCase(), u.getUsername().toUpperCase()));
}
```

1

```
public class User {
   public final String firstName;
   public final String lastName;
   public final String username;

   public User(String firstName, String lastName, String username) {
       this.firstName = firstName;
       this.lastName = lastName;
       this.username = username;
   }

   public String getFirstName() {
       return firstName;
   }

   public String getLastName() {
       return lastName;
   }

   public String getUsername() {
       return username;
   }
}
```

...

Merging in Reactor

Stage 1 - requirements:

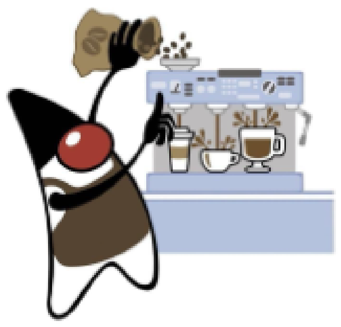
Use these two lists of strings to create some input for the methods.

```
private static List<String> words1 = Arrays.asList(
       "alpha", "bravo", "charlie", "delta", "echo", "foxtrot");

private static List<String> words2 = Arrays.asList(
       "the",
       "quick",
       "brown",
       "fox",
       "jumped",
       "over",
       "the",
       "lazy",
       "dog"
);

// TODO Merge flux1 and flux2 values with interleave
// An interesting thing to note is that, opposed to concat (lazy subscription), the
sources are subscribed eagerly.
static Flux<String> mergeFluxWithInterleave(Flux<String> flux1, Flux<String> flux2) {
   return null;
}

// TODO Merge flux1 and flux2 values with no interleave (flux1 values and then flux2
values)
static Flux<String> mergeFluxWithNoInterleave(Flux<String> flux1, Flux<String> flux2)
{
   return null;
}
```

1

```
// TODO Create a Flux containing the value of mono1 then the value of mono2
static Flux<String> createFluxFromMultipleMono(Mono<String> mono1, Mono<String>
mono2) {
   return null;
}
```

…



Reactive Producers

Stage 1 - requirements:

Use these two lists of strings to create some input for the methods.

```
private static List<String> words1 = Arrays.asList(
        "alpha", "bravo", "charlie", "delta", "echo", "foxtrot");

private static List<String> words2 = Arrays.asList(
        "the",
        "quick",
        "brown",
        "fox",
        "jumped",
        "over",
        "the",
        "lazy",
        "dog"
);

// TODO Merge flux1 and flux2 values with interleave
// An interesting thing to note is that, opposed to concat (lazy subscription), the
sources are subscribed eagerly.
static Flux<String> mergeFluxWithInterleave(Flux<String> flux1, Flux<String> flux2) {
   return null;
}

// TODO Merge flux1 and flux2 values with no interleave (flux1 values and then flux2
values)
static Flux<String> mergeFluxWithNoInterleave(Flux<String> flux1, Flux<String> flux2)
{
   return null;
}
```

```
// TODO Create a Flux containing the value of mono1 then the value of mono2
static Flux<String> createFluxFromMultipleMono(Mono<String> mono1, Mono<String>
mono2) {
   return null;
}
```