TEL-RAN

Java Course-Part 1

Dr. Yuri Granovsky

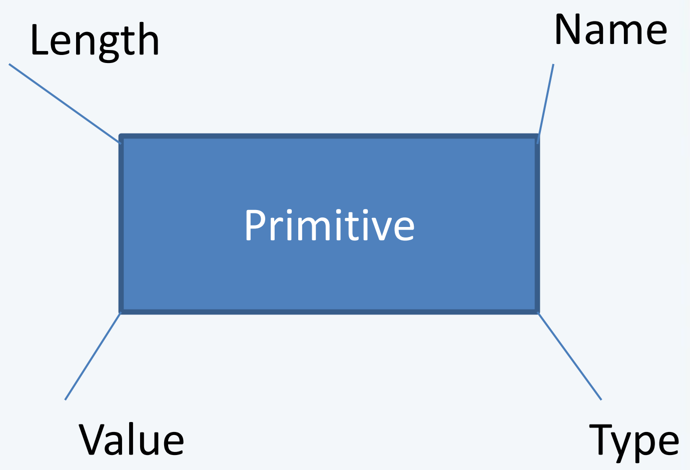
Content

- Basics
- Arrays
- Model View Controller Paradigm
- Java Collections Framework
- IP Networking (IP and Transport Layers)

Primitives

- Primitive types

Type	Length (bits)	Default Value
byte	8	0
short	16	0
int	32	0
long	64	0L
char	16	0
float	32	0.0f
double	64	0.0d
boolean	8	false



Negative Numbers

- Enumerating of bits begins from right (bit with number 0 – least significant bit) to left (bit with number ***N***-1 – most significant bit)
- Negative numbers contain 1 in the most significant bit
  - ***-1*** – all bits equaled to 1

Primitive - Operators

- Each operator returns primitive temporary value that can’t be at left side of the operator assignment
- For each type there are proper operators
  - All types except ***boolean*** support conversion operator (casting) for converting one primitive type to another
    - Java may apply casting in implicit form per rule of the saving data
  - All types except ***boolean*** support arithmetic operators (+, -, *, /, % - remainder)
  - All types except ***boolean*** support comparing operators (<, <=, >,>=,==)
  - All types except ***boolean*** support bitwise operators (~,&, ^, |, <<, >>, >>>)
    - However bitwise operators Java performs only for primitive types ***int*** and ***long*** so for other types it applies implicit casting
  - ***boolean*** type supports logical operations (&&-and, ||-or)
- Operator assignment (=, +=, /=, -=, %=, &=, ^=, <<=, >>=, >>>=)
- Increment / Decrement (++, --, postfix, prefix)

Exercise – Printing minimal and maximal values

- Write application printing minimal and maximal values for ***byte, short, int, long, char***
  - Type ***char*** doesn't have negative values so the minimal value is 0 but for other types a minimal value should be negative one computing by the following formula

  min_value= -max_value-1;

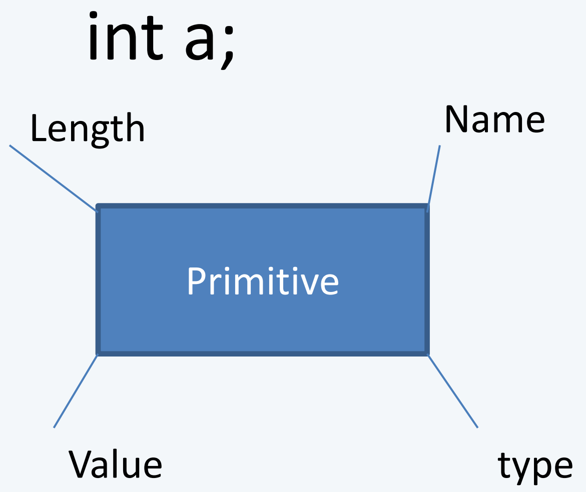
- Write intermediate function ***pow2(int k)*** allowing raising 2 to power k

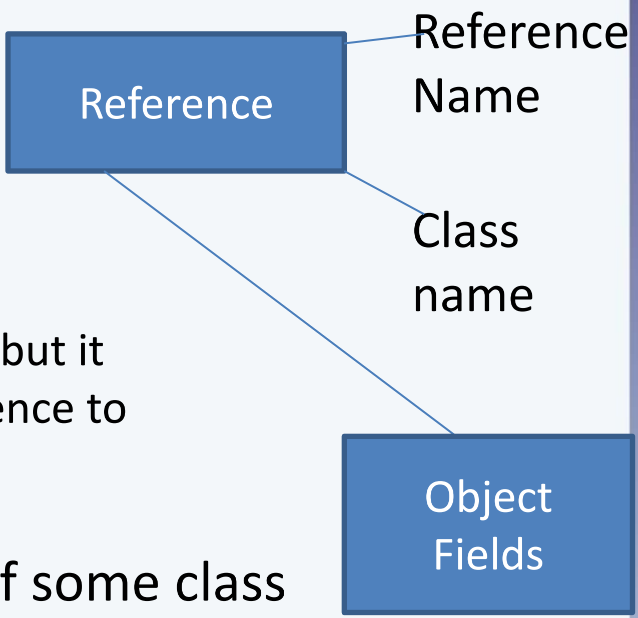
pow2(4) -> 16

Class

- Class provides specific functionality
  - String gives functionality for work with text
  - Calendar gives functionality for work with dates and time
- Class contains public fields which may be accessible from outside
  - The public fields may be :
    - Primitive constants
    - Reference to object
    - Method (any function)

Object vs. Primitive





String str;

str is not object but it is name of reference to object

- Object presents concrete instance of some class
  - Concrete string
  - Concrete date
  - Each class gives methods for object creation
    - Constructor – method with the same name as the class name (There may be several constructors in a class)
    - Some method of a class returning new instance (object) of the class

Class Usage

- Each standard class has Java documentation which is accessible from any Java IDE (for example Eclipse)
- Class has static and non-static public fields
  - Access to static fields is carried out through a class name

  <class name>.<primitive constant>|<function name>([parameters])|<reference to object>

  - Access to non-static fields is carried out through an object reference

  <reference name>.<primitive constant>|<function name>([parameters])|<reference to object>

Wrapper Classes

- For each primitive there is wrapper class encapsulating an appropriate primitive

Primitive	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Exercise – Maximal and Minimal values using Wrapper Classes

- Write function which may be called from a main function of the application class

public static double getMaxValue (String str) { }

str is reference to string containing type of primitive, for example “int”

The function shall return the maximal value of a given primitive type by using of the appropriate wrapper class, for example for “int” there should be used class Integer.

Exercise – Printing string characters by lines

- Write function which may be called from a main function of the application class

public static void printCharacters (String str) { }

  – This function shall print out characters of the string such way that each character is printed on a separate line

Algorithm Complexity- First Round getMaxLong

- Linear complexity– ***O[N]***

```
long tmp=1, max=1;
while(++tmp > 0 )
        max=tmp;
```

→ Sequential search

- Logarithmic complexity – ***O[logN]***

```
long max=1;
int N=63;
for(int i=0; i<N; i++)
        max=max*2;
```

→ Binary search

- Constant complexity – ***O[1]***

```
long max=Long.MAX_VALUE;
```

→ Direct search

String vs. StringBuffer

- Common
  - Both classes intended for keeping and processing text
- Difference
  - String is a ***immutable*** class
    - In the case a String object shall be updated a new string will be created
  - StringBuffer is ***mutable*** class
    - In the case a StringBuffer object shall be updated a new string will not be created and ***this*** object becomes updated
- Usage
  - String
    - whenever just few updates are required
    - Inner field of a class (for example, Person name)
  - StringBuffer is used whenever a lot of updates is implied

TEL-RAN

Interface

- Unlike class an interface may contain only static primitive constants and function declarations (prototypes)
- Interface can’t instantiate an object
- Java uses interfaces very often for creation of universal applications incorporating classes with common interfaces
- Class implementing an interface will add specific fields and function implementations (definitions)
  - If a class doesn’t implements all functions it is called abstract class
  - Abstract class like an interface can’t instantiate an object

Date and Calendar

- Date is an Java utilities class presenting a date and a time
  - Comparing dates
  - Converting into string
- Calendar is an Java utilities interface allowing operations on the dates such as
  - Getting any fields of a date
  - Setting any fields of a date
- GregorianCalendar is an Java class implementing Calendar interface for Gregorian calendar (365.25 days in an year, 24 hours in a day)

Date Format

- DateFormat is an Java interface for presenting date in different text formats
- SimpleDateFormat is an Java class implementing DateFormat interface with the following format
  - yy – two last digits of an year
  - yyyy – full year value
  - MM – months
  - dd – days
  - HH – hours
  - mm – minutes
- DateFormat has method ***format(<Date object>)*** that returns string presentation of a given date according to a format defined during ***SimpleDateFormat*** object creation

DateFormat df=new SimpleDateFormat("dd/MM/yyyy");

System.out.println(df.format(new Date()));

- ***DateFormat*** has method ***parse(String str)*** allowing us to get ***Date*** object from string

Exercise – Date Operations

- Write functions computing new dates per operations adding and subtracting

Date getDateAfterDays(Date date, int days)

- – Returns new date after adding ***days*** number of the days to ***date***

Date getDateAfterWeeks(Date date, int weeks)

- – Returns new date after adding ***weeks*** number of the weeks to ***date***

Date getDateBeforeMonths(Date date, int months)

- – Returns new date after subtracting ***months*** number of the months from ***date***

Exercise – Printing Hebrew Day of Week

- Write function printing date in the pointed format with Hebrew day of week

void printDateHebrewDay(Date date, DateFormat df)

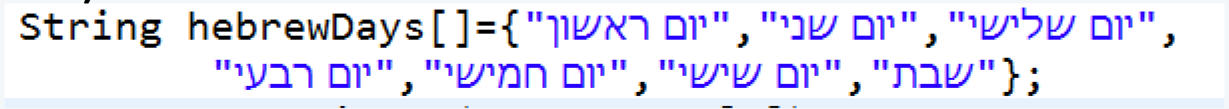
  - If ***df*** is ***null*** to print date without DateFormat object as pointed below. Note: To use StringBuffer class for the string presentation (method ***replace)***

יום שלישי Nov 19 15:41:57 IST 2013

  - If ***df*** is not ***null*** to print date in the appropriate format. For example, if format is ***“dd/MM/yy”,*** the function will print the follow

יום שלישי 11/19/13

- To use the following array (code should be saved in UTF-8 format)



```
String hebrewDays[]={"יום ראשון","יום שני","יום שלישי",
        "יום רביעי","יום חמישי","יום שישי","שבת"};
```

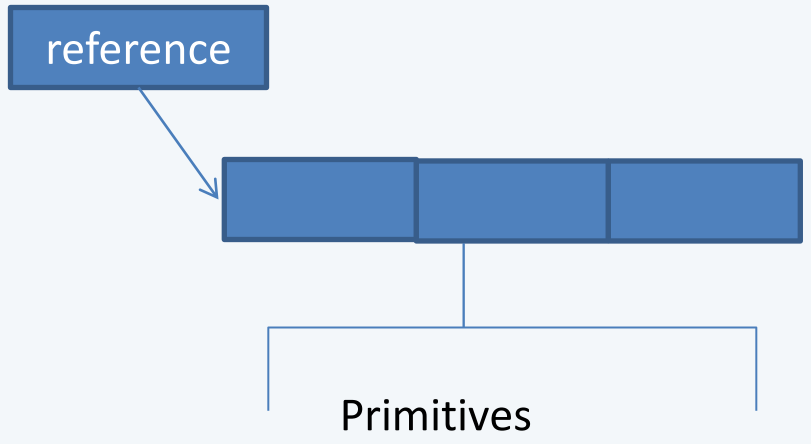
Exercise – Print Calendar

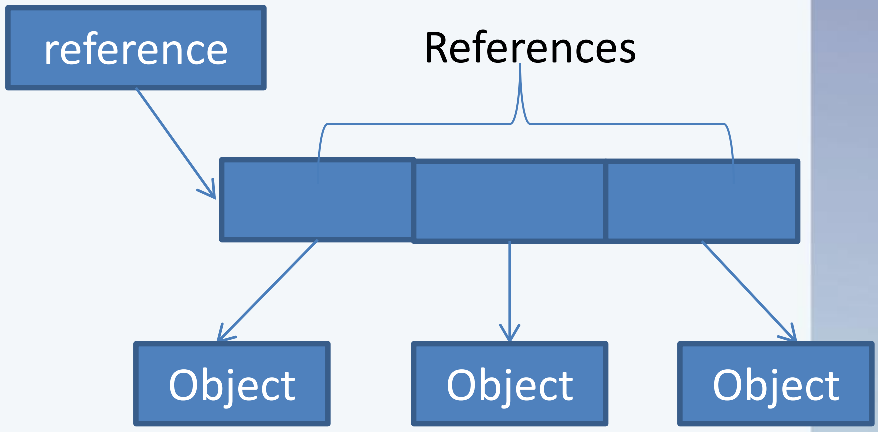
- Write function printing calendar for a given year
  - void printCalendar(int year);
  - Printing shall be in the following format

```
 <month name>
<week day name>  <day of month>….<day of month>
…………………………………………………………………………….
<week day name>  <day of month>….<day of month>
……………………………..
<month name>
<week day name>  <day of month>….<day of month>
…………………………………………………………………………….
<week day name>  <day of month>….<day of month>
```

Arrays

- Overview
  - Array is a sequence of elements which may be either primitives or references to objects
  - Array is defined by a reference to object
- Operations
  - Operator [] allowing access to an array element
  - Access to a field ***length*** allowing us to get number of the array elements (***length*** is not function so it may be used in loops)





Array Class

- Array may be presented as class with name

  <primitive>|<class> []

For example:

int [] – array of integer primitives

String[] – array of references to strings

Date[] – array of references to objects of class Date

Array Definitions

- Just reference - <array reference definition>

  <primitive>|<class> <array name> [];

  <primitive>|<class>[] <array name>;
- Reference with initialization

  <array reference definition> = {<value>,…}

  <value> may be either a primitive value or reference to object
- Array object creation

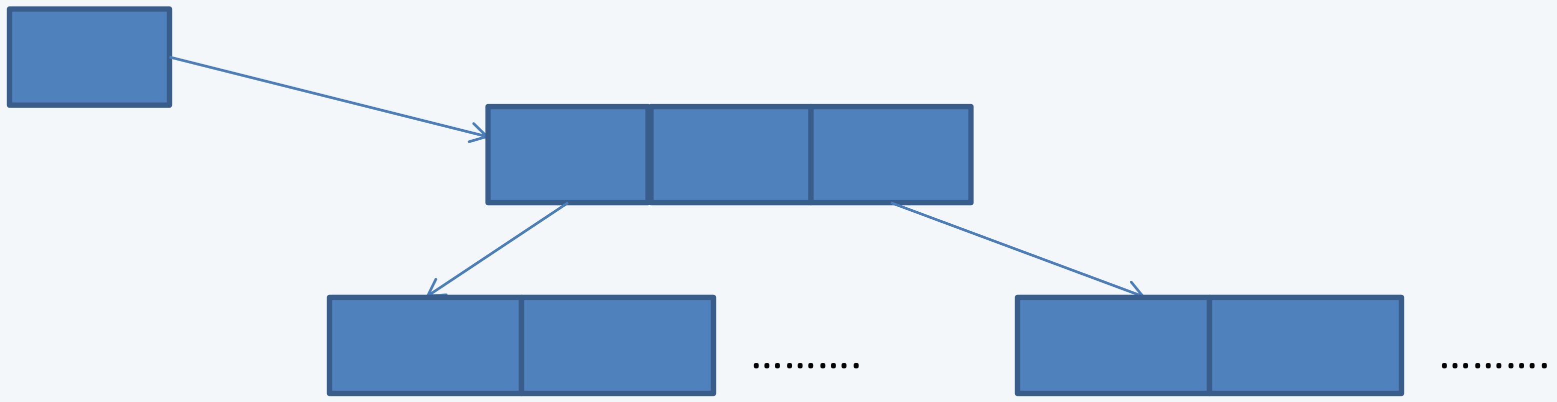
  <array reference definition> =

  new <array class>[<expression>];

Two dimensional Arrays

- Array of arrays
  - The same as one dimensional array

```
int ar[][]=new int [5][];
int ar1[][]=new int[4][10];
```



Exercise-Array definitions

- Point numbers of the right statements

1. int ar[10];
2. String ar[]=new String[10];
3. int ar[]={1,2,3};
4. String[] ar;
5. int ar[][];
6. String ar[]=new int[50];

TEL-RAN

Arrays Utilities

- Class ***Arrays*** with static methods allowing work with arrays
  - Sort

  sort(<array reference>)

  Sorts array in the ascending order

  - Search (Array should be sorted)

  binarySearch(<array reference>, <key>)

  Returns index of element matching the key if it exists otherwise (-(insert position)-1)

  - Copy

  copyOf(<array reference>, <size of new array>)

  Returns new created array with ***<size of new array>*** elements with copied elements from ***<array reference>***

  copyOfRange (<array reference>, <index from>, <index to>)

- Class ***System*** with static method ***arraycopy(<array reference source> ,<position source>,< array reference destination> ,<position destination>, <length >***

Exercise – Array Functions

- Write function converting array of strings into one string

String arrayToString(String[] array)

  - The function receives array of strings and creates one aggregated string from the strings

- Write function inserting new element into sorted array if the element doesn't exist
  - Insertion should be done in a proper position to keep on sorting

int [] insertElement(int [] sortedArr, int number)

  - Returns the same array if the array doesn't change and new array in the case of the insertion

Exercise – Matrix of week days

- Write function

int [][] matrixOfWeekDays (int year)

  - The function returns two dimensional array first index of which defines number of month , second index defines day of month and value is day of week

For example,

int [][] weekDays;

weekDays=matrixOfWeekDays(2013);

weekDays[10][26] -> 3

TEL-RAN

Java Class

- Syntax

[public] [final]class <name> [extends <name>] [implements <name>, …] <class definition>

<class definition>:={<field definition>…}

<field definition>:=[<access>][static][final] <data member >|<method>

<data member>:=<primitive> | <class name> <name>;

<***method***>:= <primitive>|<class name>| void <function name>

([<parameter definition>,…]) {<function body>}

<***access***>:=private|public|protected

Constructor

- Syntax

```
<class name> ([<parameter definition>,...])
{ <function body>}
```

- Constructor call

```
new <class name>([parameter value,...])
```

  - Operator ***new*** returns reference to object of the <class name> class
  - If there is no a constructor the default constructor is implied
  - If there is/are constructor/s an object may be created only by using existing constructors

Interface

- Contains only constants and method declarations

[public] interface <name> [extends <interfaces name>] { <interface definition>}

<***interface definition***>:=[public]<constant> | <function declaration>

<***constant***>:=static final <primitive> | <reference> = <value definition>

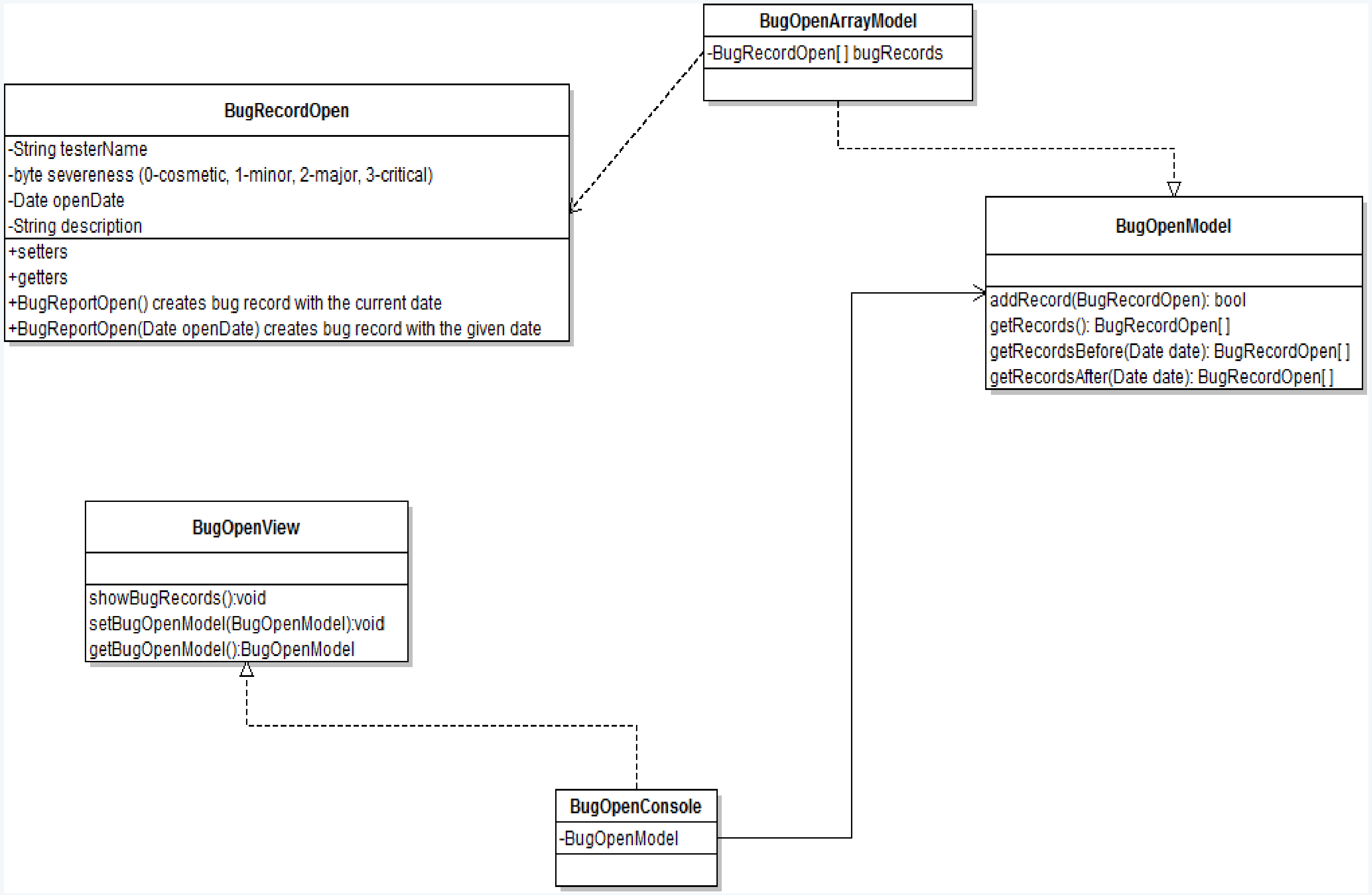
<***method declaration***>:= <primitive>|<class name>| void <function name>

([<parameter definition>,…]);

Model View Controller -OOP Paradigm

- Model
  - Information presentation
- View
  - Human presentation
- Controller
  - Application creating and managing objects

Exercise-BugOpenRecord


BugOpenArrayModel
-BugRecordOpen[ ] bugRecords
BugRecordOpen
-String testerName
-byte severeness (0-cosmetic, 1-minor, 2-major, 3-critical)
-Date openDate
-String description
+setters
+getters
+BugReportOpen() creates bug record with the current date
+BugReportOpen(Date openDate) creates bug record with the given date
BugOpenModel
addRecord(BugRecordOpen): bool
getRecords(): BugRecordOpen[ ]
getRecordsBefore(Date date): BugRecordOpen[ ]
getRecordsAfter(Date date): BugRecordOpen[ ]
BugOpenView
showBugRecords():void
setBugOpenModel(BugOpenModel):void
getBugOpenModel():BugOpenModel
BugOpenConsole
-BugOpenModel

Array Sorting

- Method ***Arrays.sort(Object[] array)***
  - Class Object – class implying designation of any class
- Quick sort algorithm with complexity ***O[N*logN]***
  - Bubble sort algorithm has complexity ***O[N^2]***
- Sorting an array of the objects of any class - How is it possible ?
  - Being sorted objects must be instances of a comparable class
    - Comparable class implements interface ***Comparable***

Comparable Interface

- Standard universal interface with the method getting meaning of the comparing

public int compareTo (Object obj);

  - Class Object – class implying designation of any class
  - Implementation of this method:
    - should contain casting to the concrete type
    - Returns negative number if ***this*** object less than being received, positive number if ***this*** object more than being received and zero if ***this*** object equals the being received
- Parameterization of a concrete type

Comparable< <class name> >

  - Gives possibility to implement the following method

  public int compareTo (<class name> obj);

Sorting with Comparator

- Comparator is an universal standard interface containing method of comparing

int compare(Object obj1, Object obj2) ;

  – Returns negative number if obj1 less than obj2, positive number if obj1 more than obj2 and zero if they are equaled

- Comparator parameterization

Comparator< <class name> > is parameterized interface with method

int compare(<class name> obj1, <class name> obj2);

- Array sorting with comparator

Arrays.sort(T [] arr, Comparator <T> comp)

Exercise – Bug Report Upgrade

- Update ***BugOpenModel***
  - Add new method

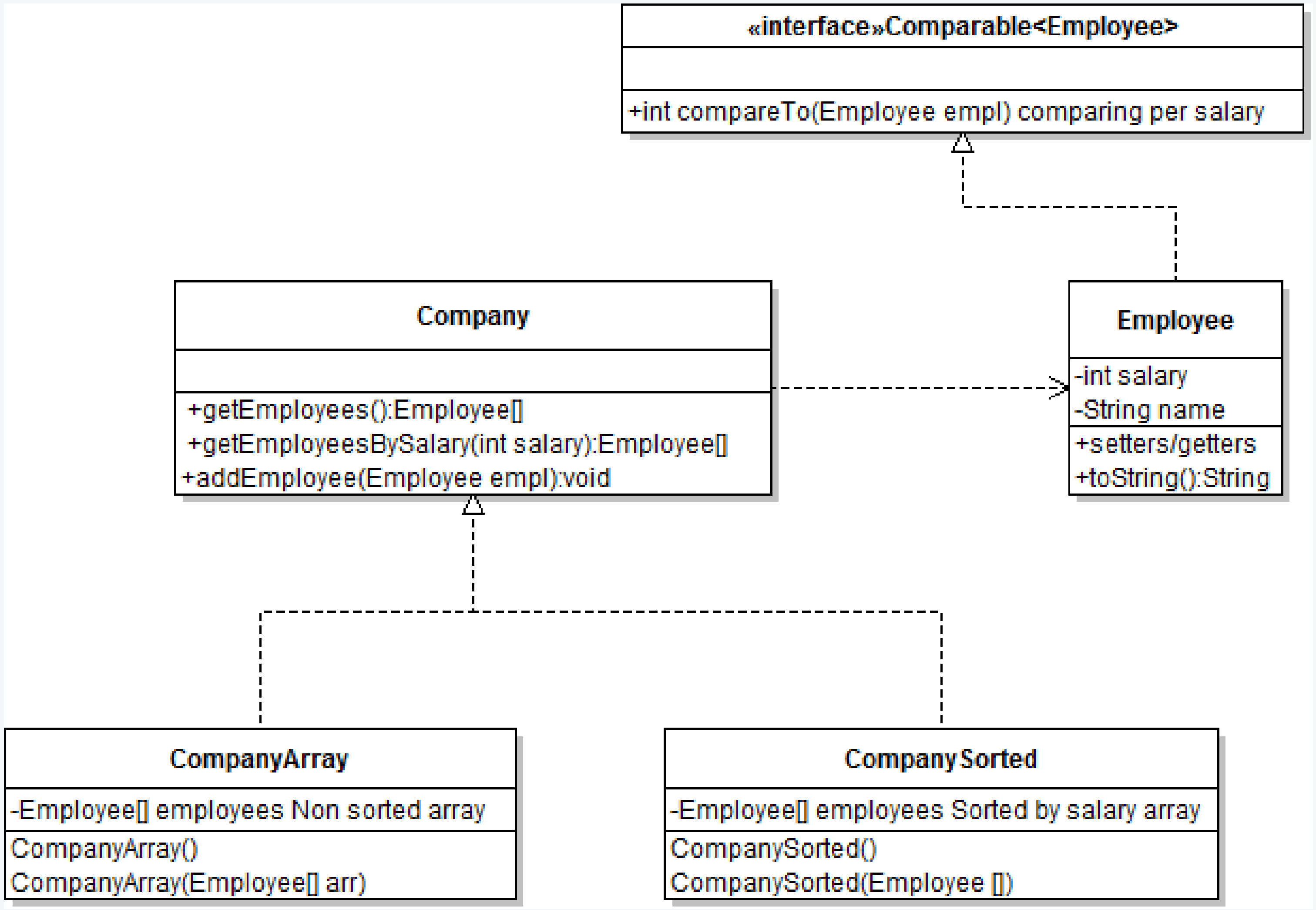
    BugRecordOpen[] getSortedRecords(Comparator<BugRecordOpen> comp);
- Update ***BugOpenArrayModel*** by implementing ***getSortedRecords***
- Update ***BugRecordOpen***
  - Implementing interface ***Comparable<BugRecordOpen>*** with method

    int compareTo(BugRecordOpen obj) allowing comparison per open dates
- Develop class ***BugOpenSortedModel*** implementing ***BugOpenModel*** interface
  - Contains array BugRecordOpen[] sorted by open date
  - Constructor BugSortedModel(BugRecordOpen[]array). Note: the constructor receives an array which may or may not be sorted

Exercise Employee-Company

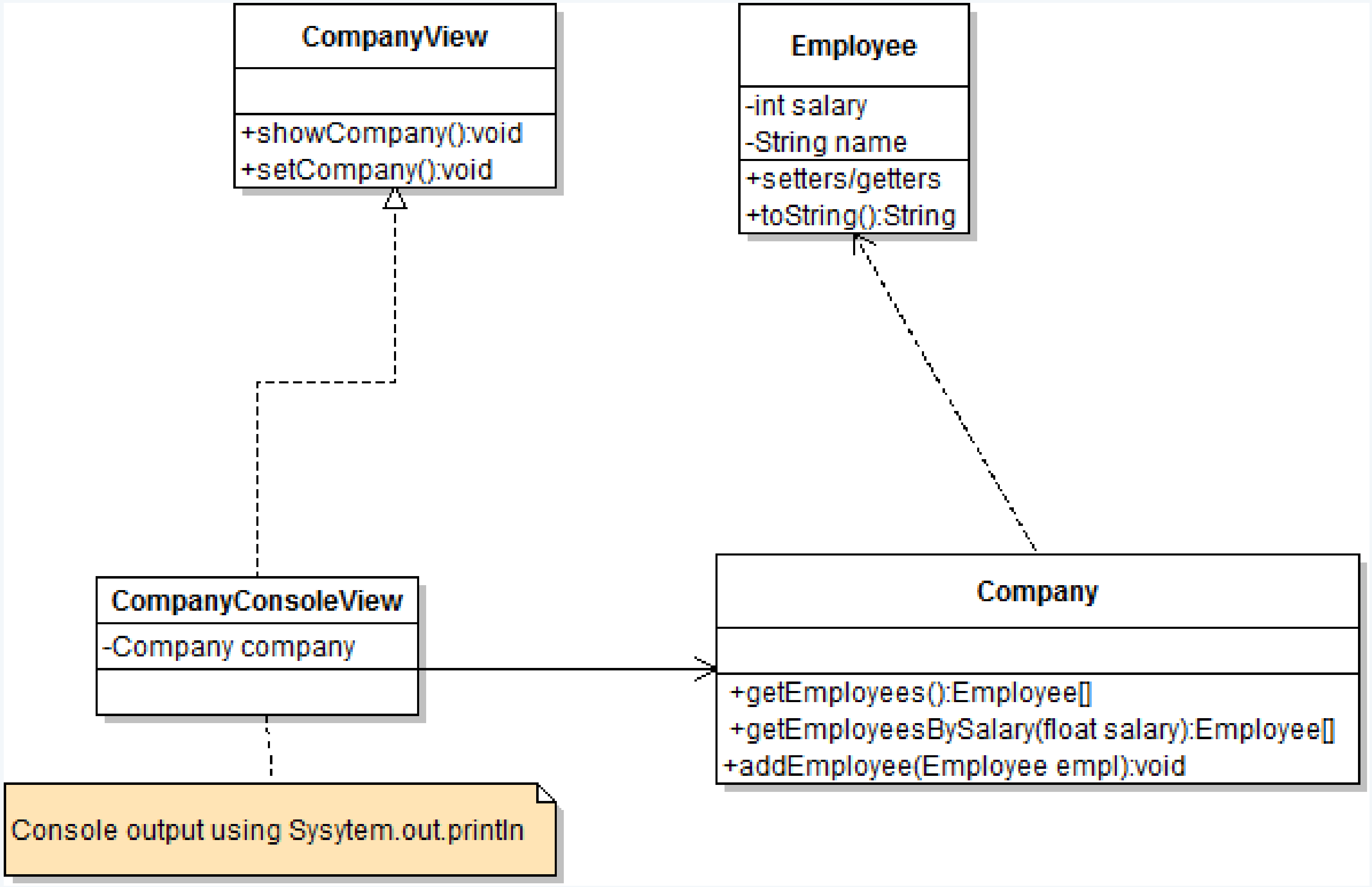
- Model (page 38)
  - Company with functionality as pointed on the UML diagram
  - Two model's implementations:
    - CompanyArray (unsorted array)
    - CompanySorted (sorted per salary array)
- View (page 39)
  - CompanyView with functionality as pointed on the UML diagram
  - Console view implementation
- Controller
  - See page 40

UML Class Diagram - Company



«interface»Comparable<Employee>
+int compareTo(Employee empl) comparing per salary
Company
+getEmployees():Employee[]
+getEmployeesBySalary(int salary):Employee[]
+addEmployee(Employee empl):void
Employee
-int salary
-String name
+setters/getters
+toString():String
CompanyArray
-Employee[] employees Non sorted array
CompanyArray()
CompanyArray(Employee[] arr)
CompanySorted
-Employee[] employees Sorted by salary array
CompanySorted()
CompanySorted(Employee [])

UML Class Diagram – Company View



Employee Controller

- Create Company as CompanySorted
- Fill company with 100 random employees using Math.random() method returning a random number in the range [0, 1.0)
  - Random salary ***(int)(Math.random()*(maxSalary-minSalary+1))+minSalary***
  - Random employee name as ***names[indNames]***

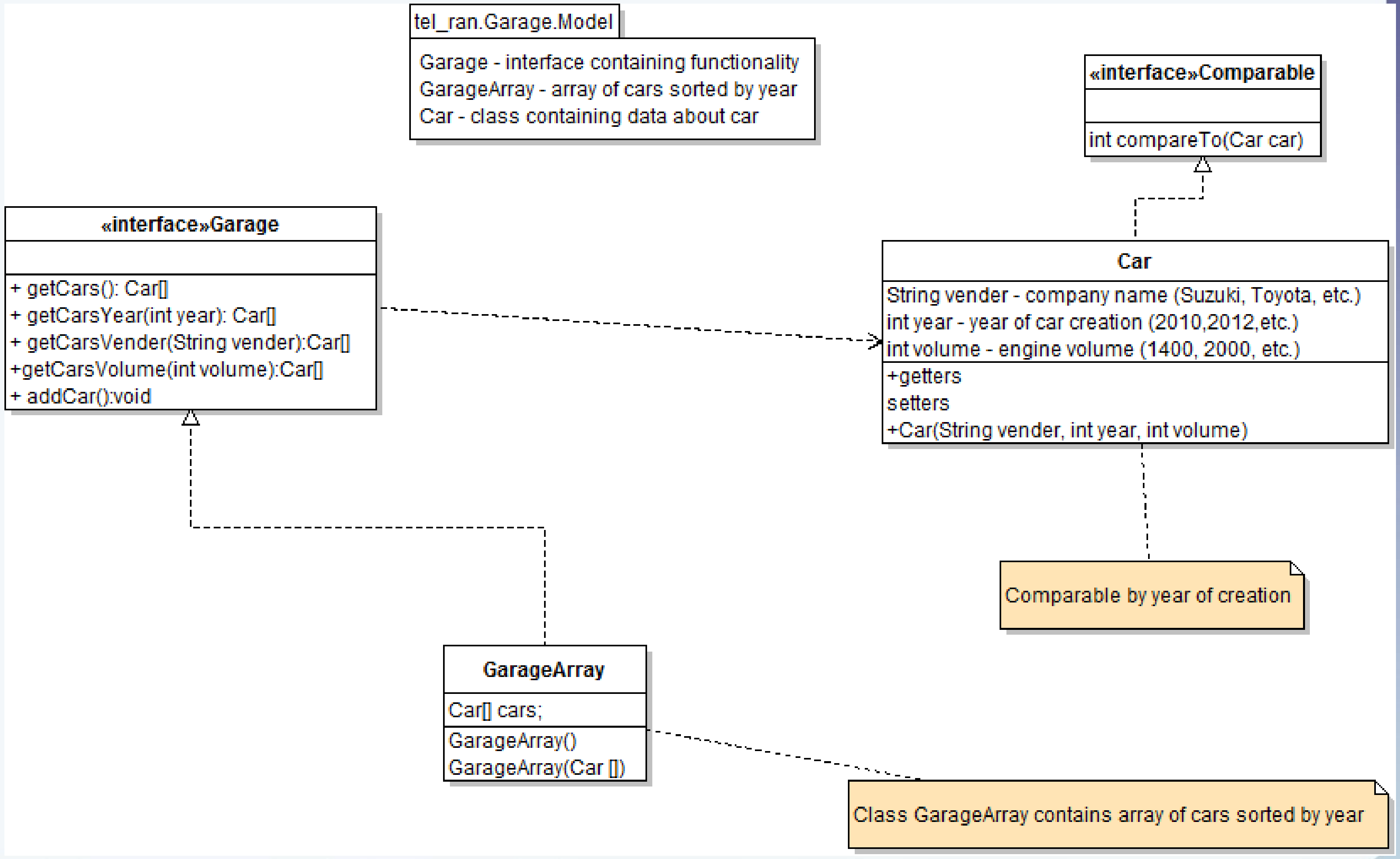
  String names={“Moshe”, “David”,…………};

  int indNames=(int)(Math.random()*names.length);
- Show all employees sorted by salary values
- Find all employees with maximal salary and show them in the order sorted by employee name values
- Find all employees with minimal salary and show them in the order sorted by employee name values

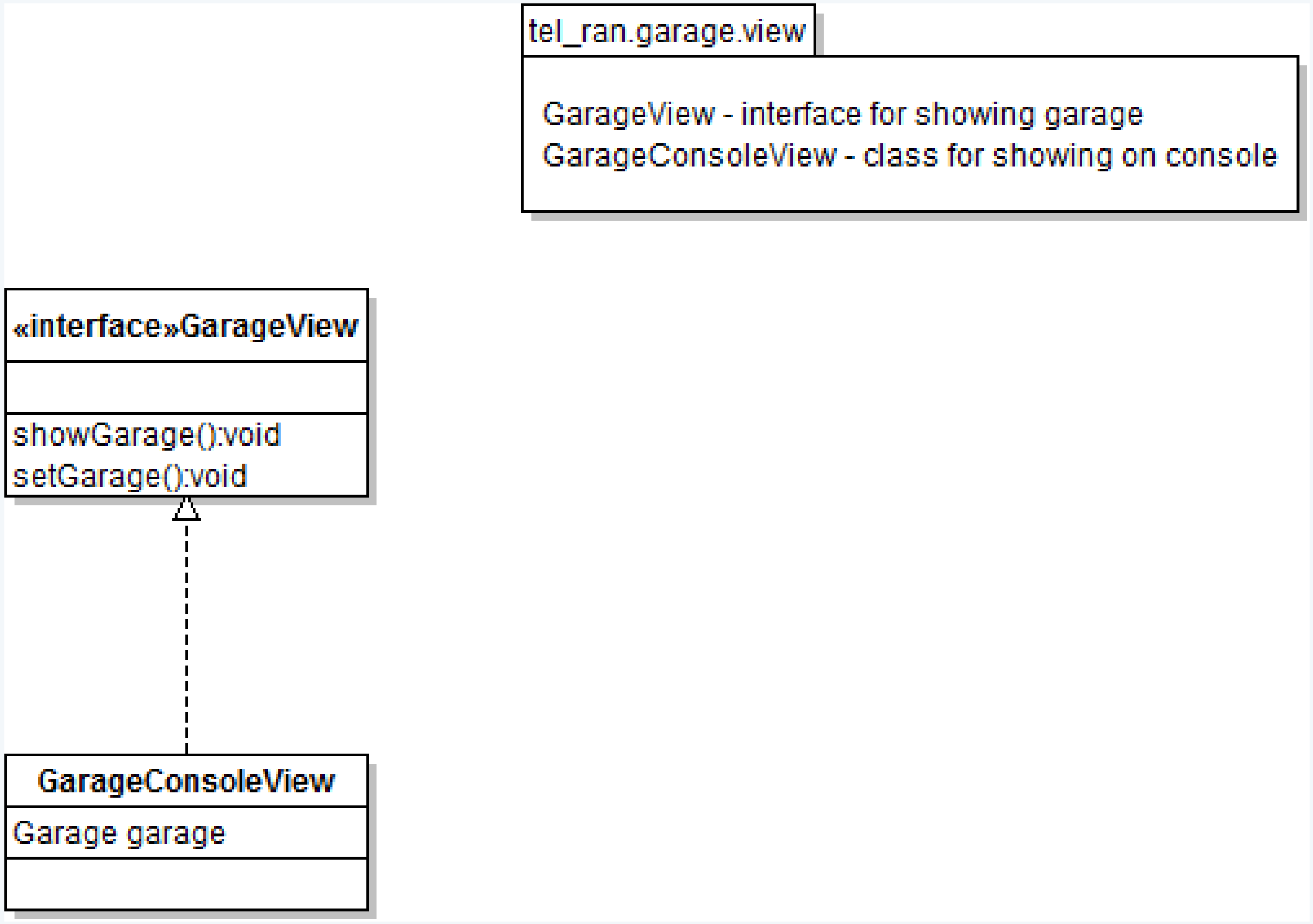
Exercise – Garage Application

- Implement UML class diagrams on the slides 42-44

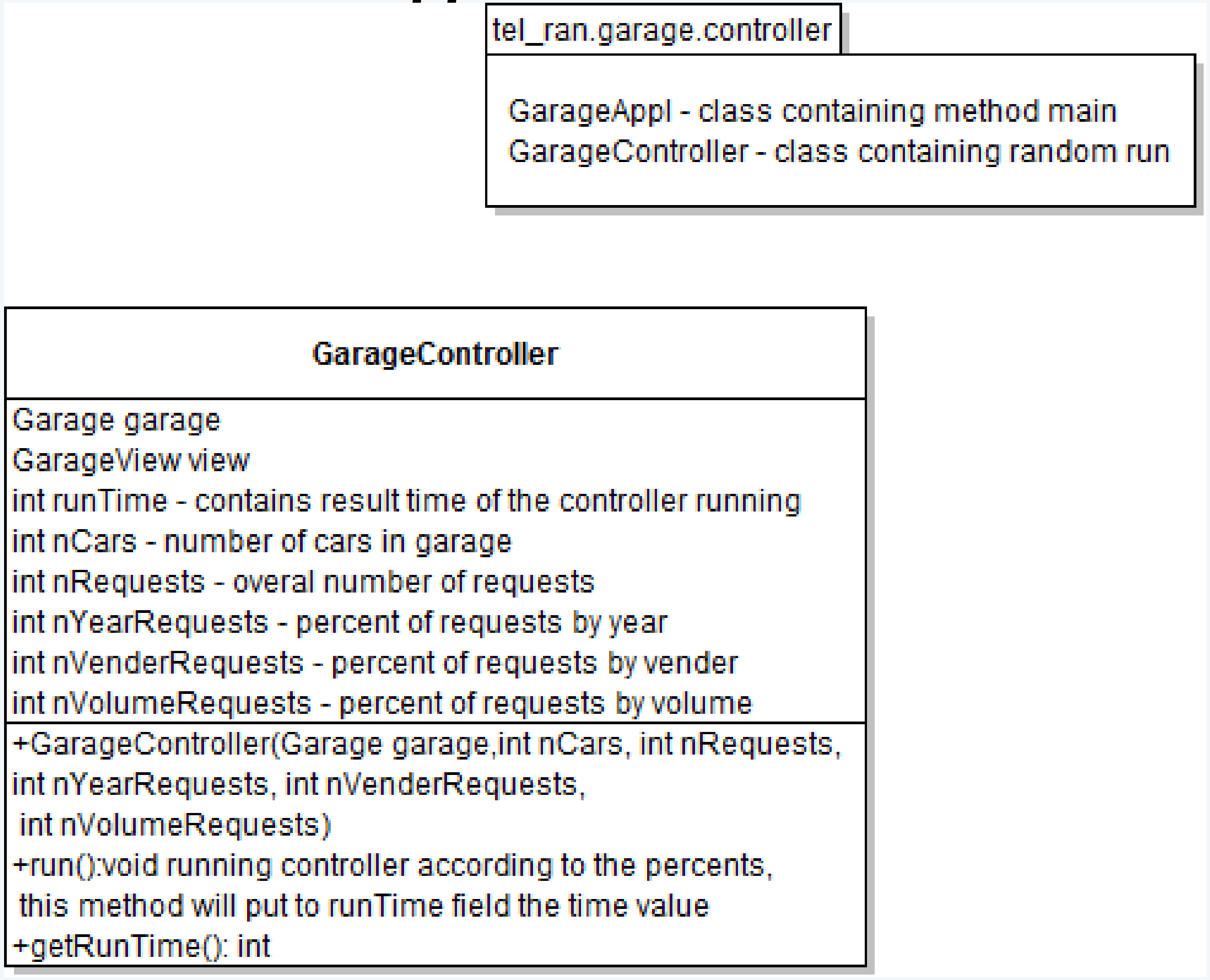
Garage Model


tel_ran.Garage.Model
Garage - interface containing functionality
GarageArray - array of cars sorted by year
Car - class containing data about car
«interface»Comparable
int compareTo(Car car)
«interface»Garage
+ getCars(): Car[]
+ getCarsYear(int year): Car[]
+ getCarsVender(String vender):Car[]
+getCarsVolume(int volume):Car[]
+ addCar():void
Car
String vender - company name (Suzuki, Toyota, etc.)
int year - year of car creation (2010,2012,etc.)
int volume - engine volume (1400, 2000, etc.)
+getters
setters
+Car(String vender, int year, int volume)
Comparable by year of creation
GarageArray
Car[] cars;
GarageArray()
GarageArray(Car [])
Class GarageArray contains array of cars sorted by year

Garage View


tel_ran.garage.view
GarageView - interface for showing garage
GarageConsoleView - class for showing on console
«interface»GarageView
showGarage():void
setGarage():void
GarageConsoleView
Garage garage

Garage Controller



tel_ran.garage.controller

GarageAppl - class containing method main
GarageController - class containing random run

GarageController

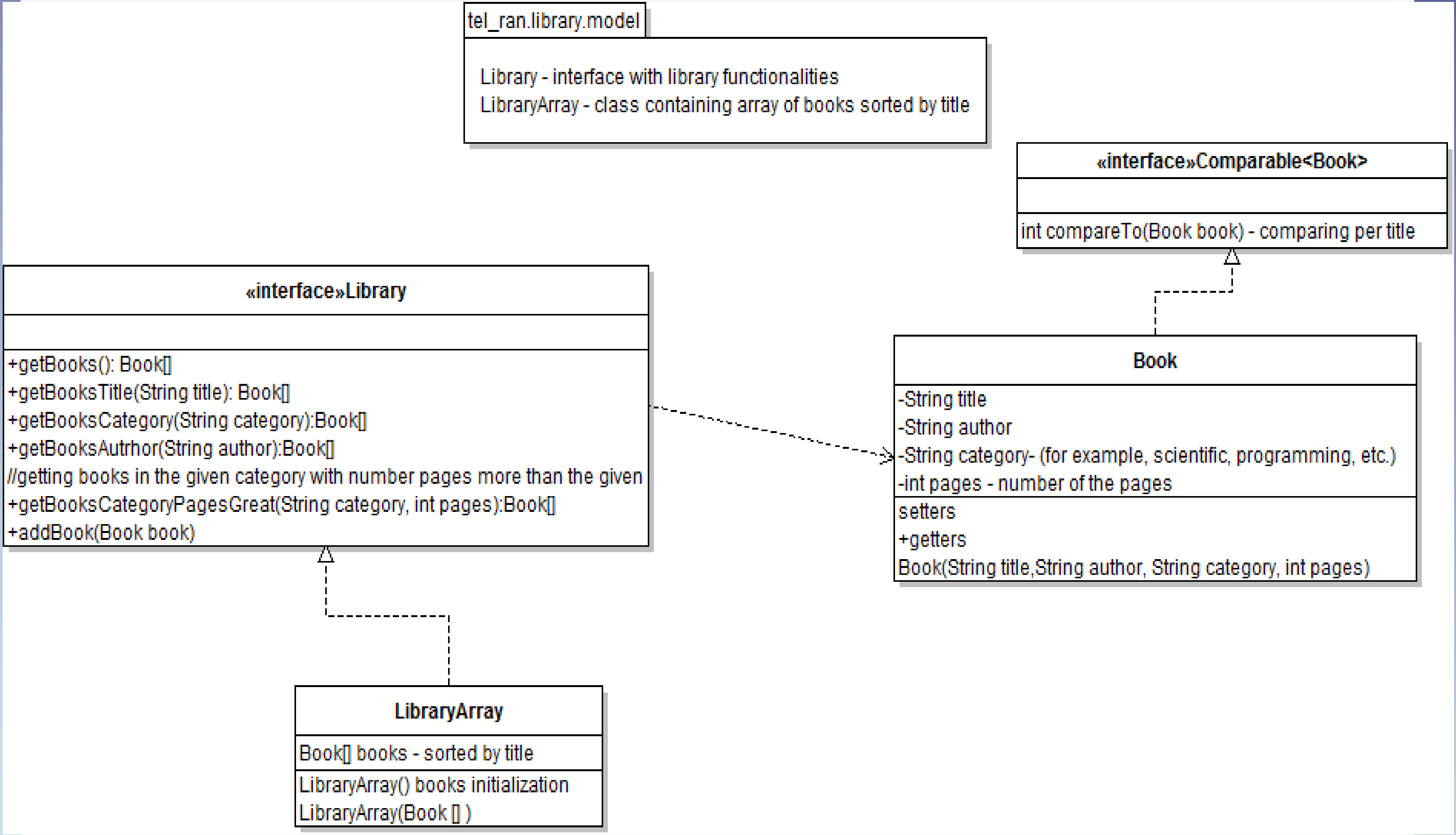
Garage garage
GarageView view
int runTime - contains result time of the controller running
int nCars - number of cars in garage
int nRequests - overal number of requests
int nYearRequests - percent of requests by year
int nVenderRequests - percent of requests by vender
int nVolumeRequests - percent of requests by volume

+GarageController(Garage garage,int nCars, int nRequests,
int nYearRequests, int nVenderRequests,
int nVolumeRequests)
+run():void running controller according to the percents,
this method will put to runTime field the time value
+getRunTime(): int

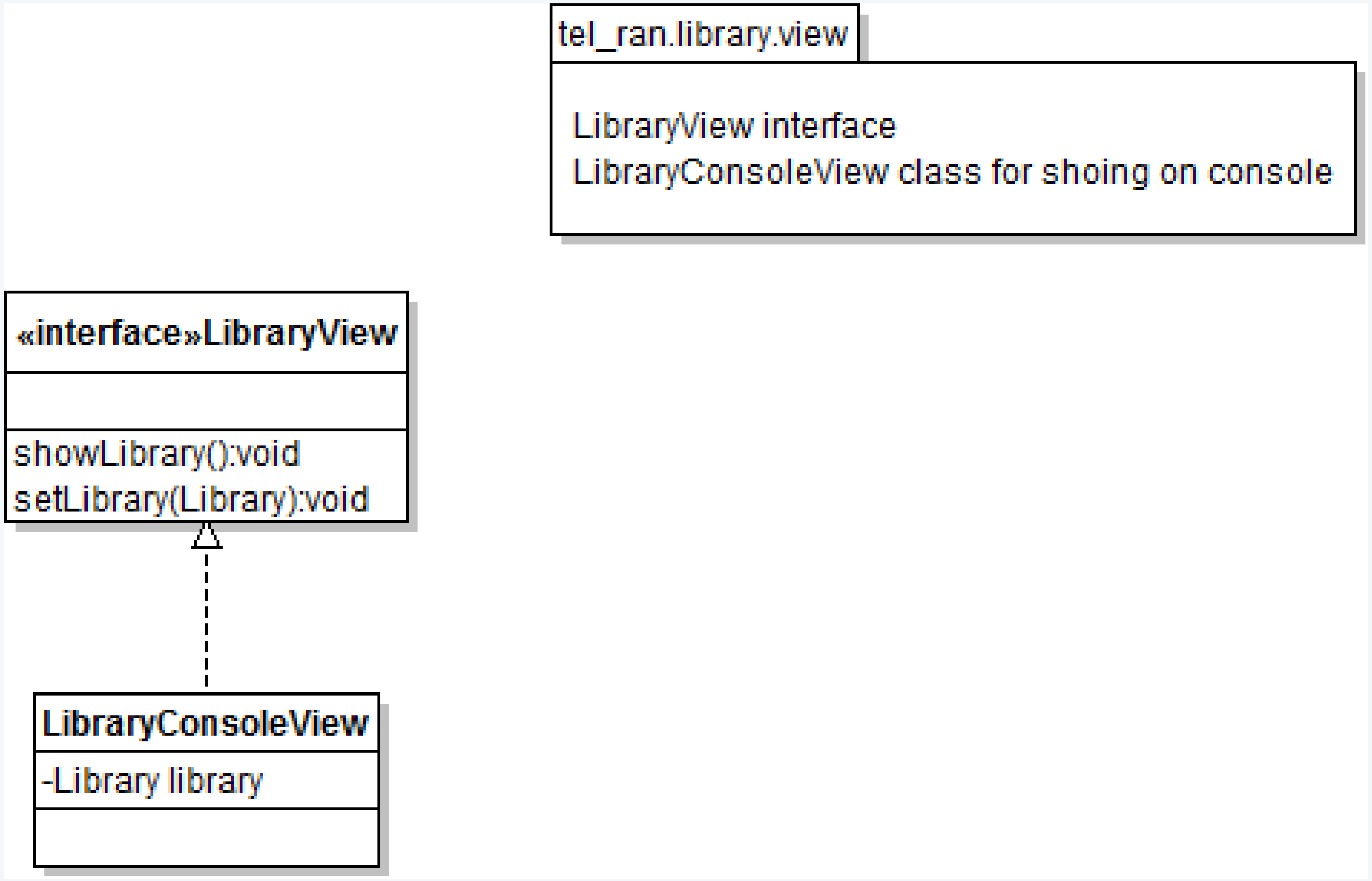
Exercise – Library Application

- Implement UML diagrams on the slides 46-48

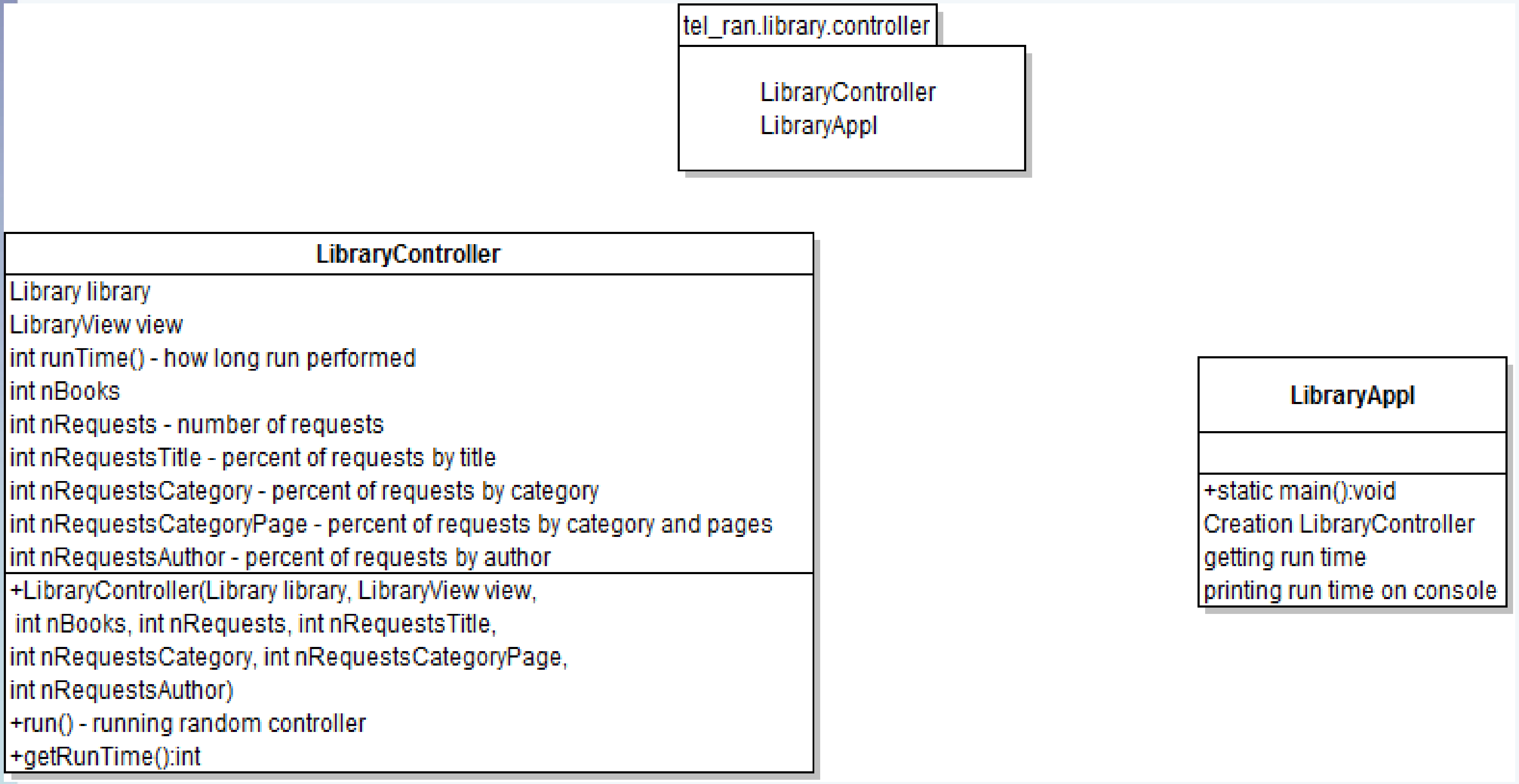
Library Model


tel_ran.library.model
Library - interface with library functionalities
LibraryArray - class containing array of books sorted by title
«interface»Comparable<Book>
int compareTo(Book book) - comparing per title
«interface»Library
+getBooks(): Book[]
+getBooksTitle(String title): Book[]
+getBooksCategory(String category):Book[]
+getBooksAutrhor(String author):Book[]
//getting books in the given category with number pages more than the given
+getBooksCategoryPagesGreat(String category, int pages):Book[]
+addBook(Book book)
Book
-String title
-String author
-String category- (for example, scientific, programming, etc.)
-int pages - number of the pages
setters
+getters
Book(String title,String author, String category, int pages)
LibraryArray
Book[] books - sorted by title
LibraryArray() books initialization
LibraryArray(Book [])

Library View



Library Controller



LibraryController

Library library
LibraryView view
int runTime() - how long run performed
int nBooks
int nRequests - number of requests
int nRequestsTitle - percent of requests by title
int nRequestsCategory - percent of requests by category
int nRequestsCategoryPage - percent of requests by category and pages
int nRequestsAuthor - percent of requests by author

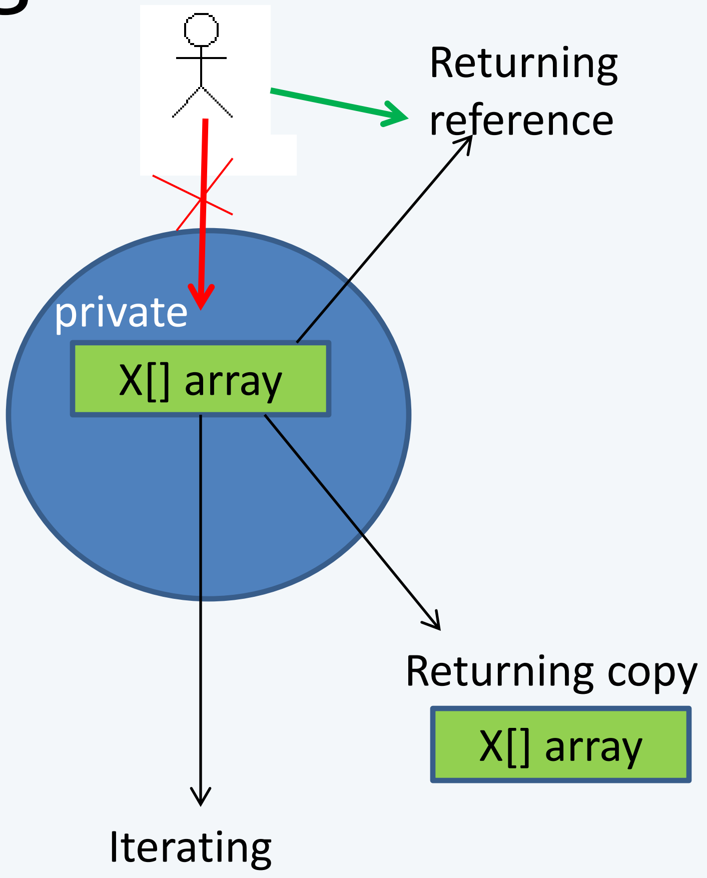
+LibraryController(Library library, LibraryView view,
 int nBooks, int nRequests, int nRequestsTitle,
int nRequestsCategory, int nRequestsCategoryPage,
int nRequestsAuthor)
+run() - running random controller
+getRunTime():int

LibraryAppl

+static main():void
Creation LibraryController
getting run time
printing run time on console

Model Getters Issues and Model Iterating

- Returning reference
  - Opening an access from outside
- Returning copy
  - Additional memory consumption
  - Performance reduction
- Iterating
  - Returning iterator object as the one allowing model iterating (Pass of a model)



Iterator Design Pattern

- Design Pattern
  - Pattern of the solution for resolving some issue under some purpose
- Iterator Pattern
  - Pattern for providing a pass of elements with encapsulating of an elements collection (array, Database, etc.)
- Iterator Interface
  - Standard Java interface ***Iterator<T>*** with following methods:

public boolean hasNext() returns ***true*** if there is element

public T next() returns next element of the type ***T*** in iteration

public void remove() removes element but we will leave it as {} empty method

Iterable Interface

- Interface with the method

Iterator<T> iterator();

- A class implementing the Iterable<T> has to implement the aforementioned method
- Iterating of an iterable object (object of a class implementing Iterable<T> interface)

for(T tObj: iterableObj)

System.out.println(tObj)

  - The ***for*** iterates some iterable object getting each ***T*** object from the object of an iterable class in each iterations

Exercise – Generator of the Random Integer Numbers

- Write class ***GenRandInt*** allowing iterating of a given amount of the random integer numbers in the given range [min, max)
  - The class shall implements interface ***Iterable<Integer>***
  - Write ***ArrayIntIterator*** class implementing interface ***Iterator<Integer>*** with the following methods

    bool hasNext() – returns true if there is next integer number for iteration

    Integer next() – returns current integer number and increases current value of an index

    void remove() – empty method (does nothing)
- Write class ***GenRand*** with static method ***main*** for testing class ***GenRandInt***
  - Iterating using for from the previous slide

Exercise – ***Library*** Component improvement

- Library Model
  - Update ***Library*** interface

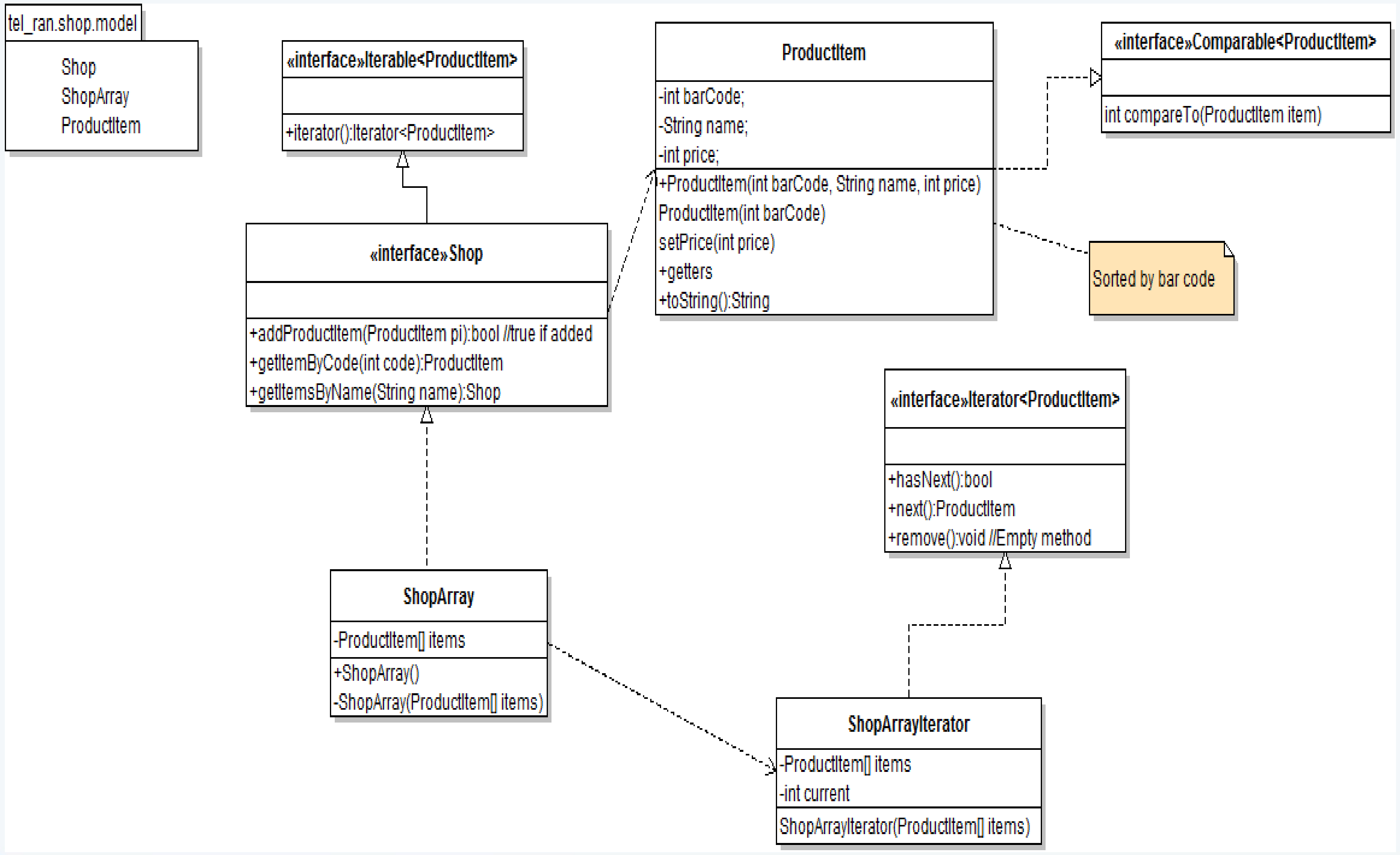
    interface Library extends Itarable<Book> this make interface ***Library*** be ***Iterable<Book>*** too

    - Remove method getBooks()
    - All methods ***get*** return reference to object of the class implementing ***Library*** interface
  - Update ***LibraryArray*** class
    - Constructor ***LibraryArray (Book[] books)*** should be private for using only from methods ***get.*** This constructor will only assign references with no copying and sorting
    - Update all functions ***get*** according to the updated interface
      - The update of functions will be only in using private constructor ***LibraryArray (Book[] books***) returning object of the class LibraryArray created from the array
    - Implement method ***iterator***() returning iterator for iterating books
      - Write class LibraryArrayIterator similar to ArrayIntIterator from previous exercise implementing interface Iterator<Book>

Exercise – Shop Application Model

- Develop Shop model
  - UML diagram (page 55)
  - Package tel_ran.shop.model
    - Class ProductItem
    - Interface Shop extending interface Iterable<ProductItem>
    - Class ShopArrayIterator implementing interface Iterator<ProductItem> for iterating array of product items
    - Class ShopArray implementing interface Shop based on array sorted by barCode

UML Diagram –Shop Model

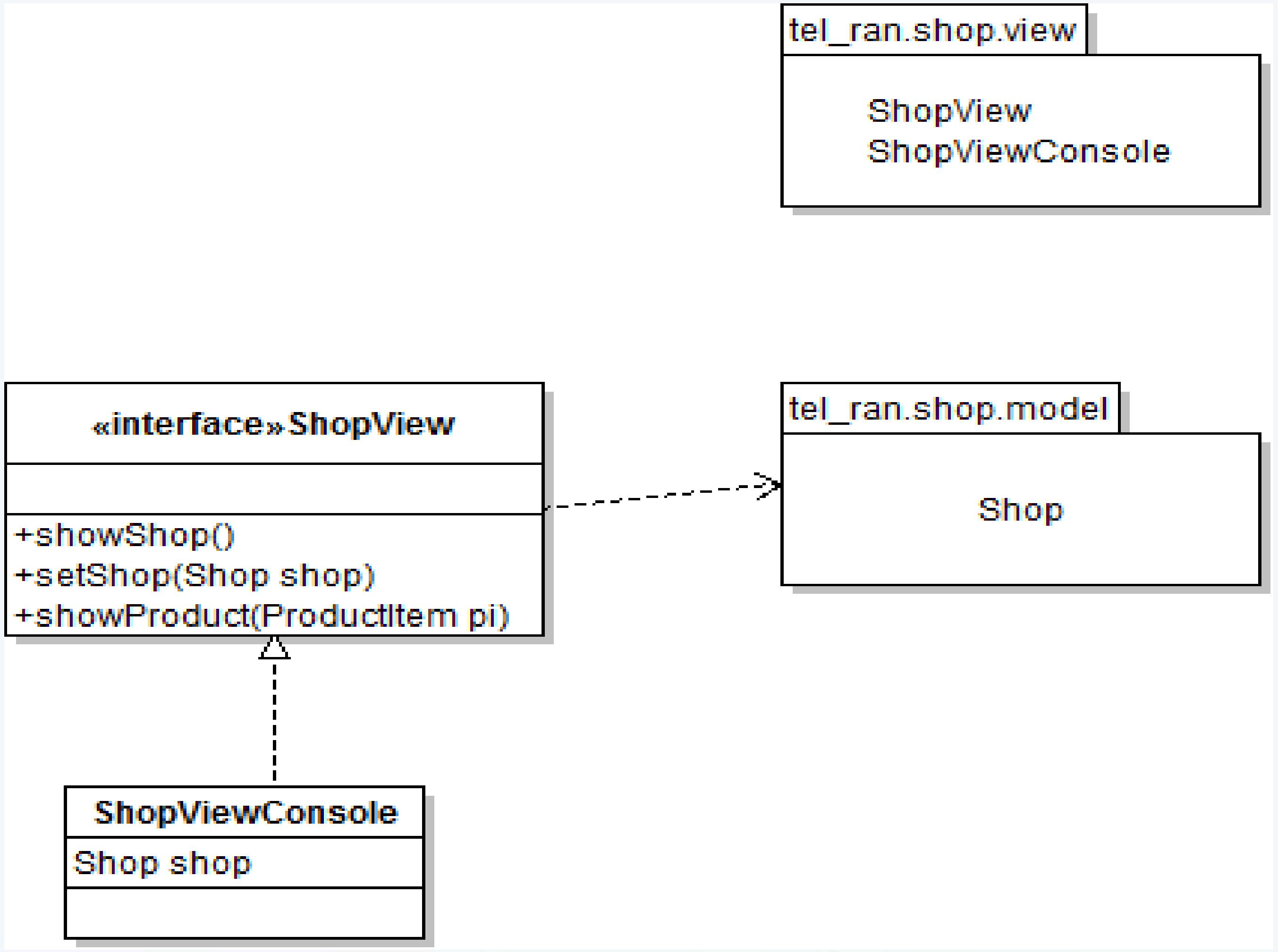


tel_ran.shop.model
Shop
ShopArray
ProductItem
«interface»Iterable<ProductItem>
+iterator():Iterator<ProductItem>
ProductItem
-int barCode;
-String name;
-int price;
+ProductItem(int barCode, String name, int price)
ProductItem(int barCode)
setPrice(int price)
+getters
+toString():String
«interface»Comparable<ProductItem>
int compareTo(ProductItem item)
Sorted by bar code
«interface»Shop
+addProductItem(ProductItem pi):bool //true if added
+getItemByCode(int code):ProductItem
+getItemsByName(String name):Shop
«interface»Iterator<ProductItem>
+hasNext():bool
+next():ProductItem
+remove():void //Empty method
ShopArray
-ProductItem[] items
+ShopArray()
-ShopArray(ProductItem[] items)
ShopArrayIterator
-ProductItem[] items
-int current
ShopArrayIterator(ProductItem[] items)

Exercise – Shop Application View

- Develop Shop view
  - UML Diagram (page 57)
  - Package tel_ran.shop.view
    - Interface ShopView
    - Class ShopViewConsole implementing interface ShopView based on System.out.println(…)

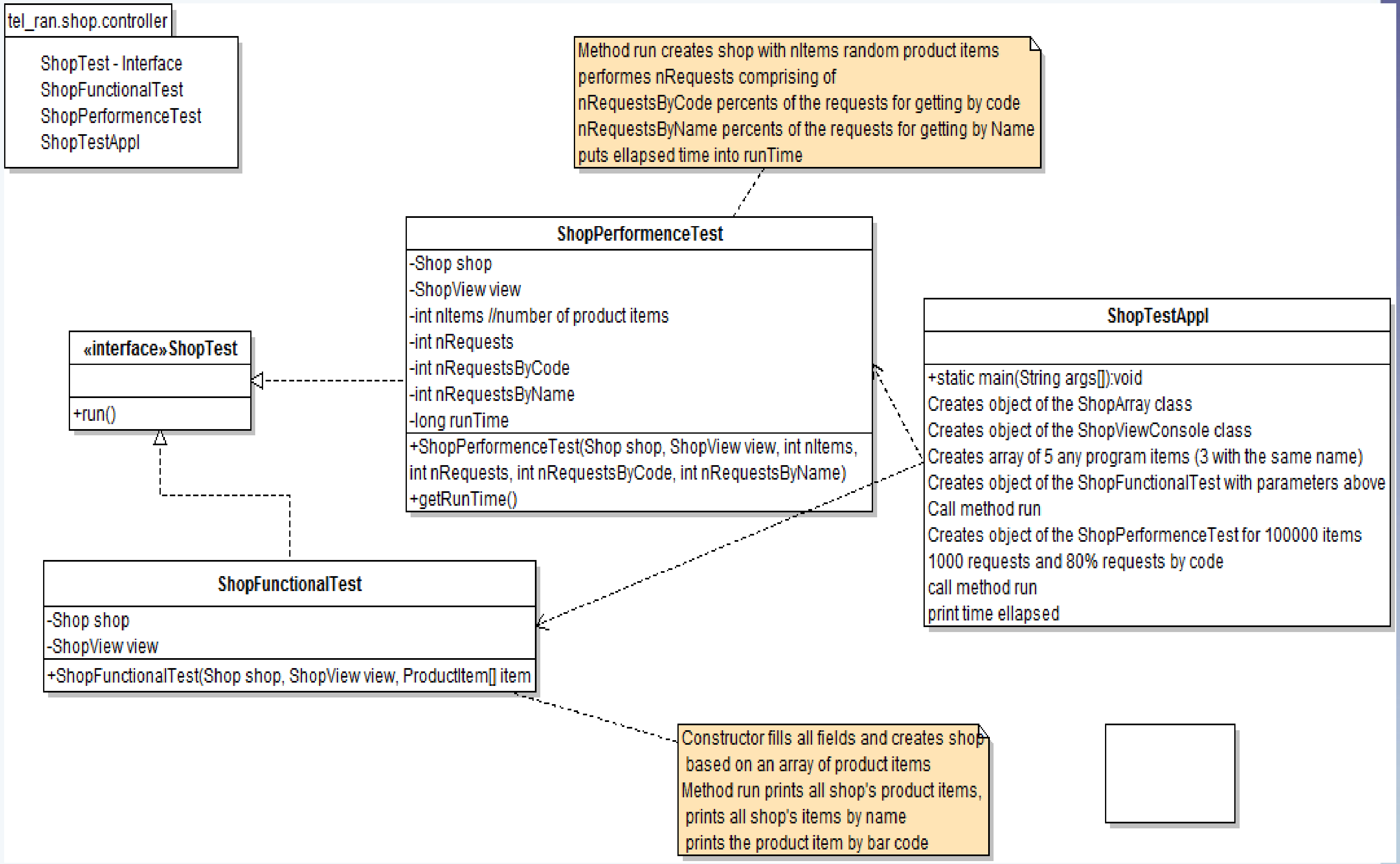
UML Diagram-Shop View



Exercise – Shop Application Controller

- Develop Shop test application
- UML Diagram (page 59)
  - Package tel_ran.shop.controller
    - Interface ShopTest
    - Class ShopFunctionalTest implementing interface ShopTest for running functional test.
      - Constructor creates Shop based on the small array of items being received from the user (main function)
      - Prints all items
      - Prints item by existing bar code
      - Prints items with a given name
    - Class ShopPerformenceTest implementing interface ShopTest
      - Constructor creates Shop containing a given number of the random created items
      - Computes time of running a given big amount of requests
        - » Given percent of the requests by bar code
        - » Given percent of the requests by a name
    - Class ShopTestAppl containing method main

UML Diagram Shop Controller



tel_ran.shop.controller

ShopTest - Interface
ShopFunctionalTest
ShopPerformenceTest
ShopTestAppl

Method run creates shop with nItems random product items
performes nRequests comprising of
nRequestsByCode percents of the requests for getting by code
nRequestsByName percents of the requests for getting by Name
puts ellapsed time into runTime

ShopPerformenceTest

-Shop shop
-ShopView view
-int nItems //number of product items
-int nRequests
-int nRequestsByCode
-int nRequestsByName
-long runTime

+ShopPerformenceTest(Shop shop, ShopView view, int nItems, int nRequests, int nRequestsByCode, int nRequestsByName)
+getRunTime()

«interface»ShopTest

+run()

ShopTestAppl

+static main(String args[]):void
Creates object of the ShopArray class
Creates object of the ShopViewConsole class
Creates array of 5 any program items (3 with the same name)
Creates object of the ShopFunctionalTest with parameters above
Call method run
Creates object of the ShopPerformenceTest for 100000 items
1000 requests and 80% requests by code
call method run
print time ellapsed

ShopFunctionalTest

-Shop shop
-ShopView view

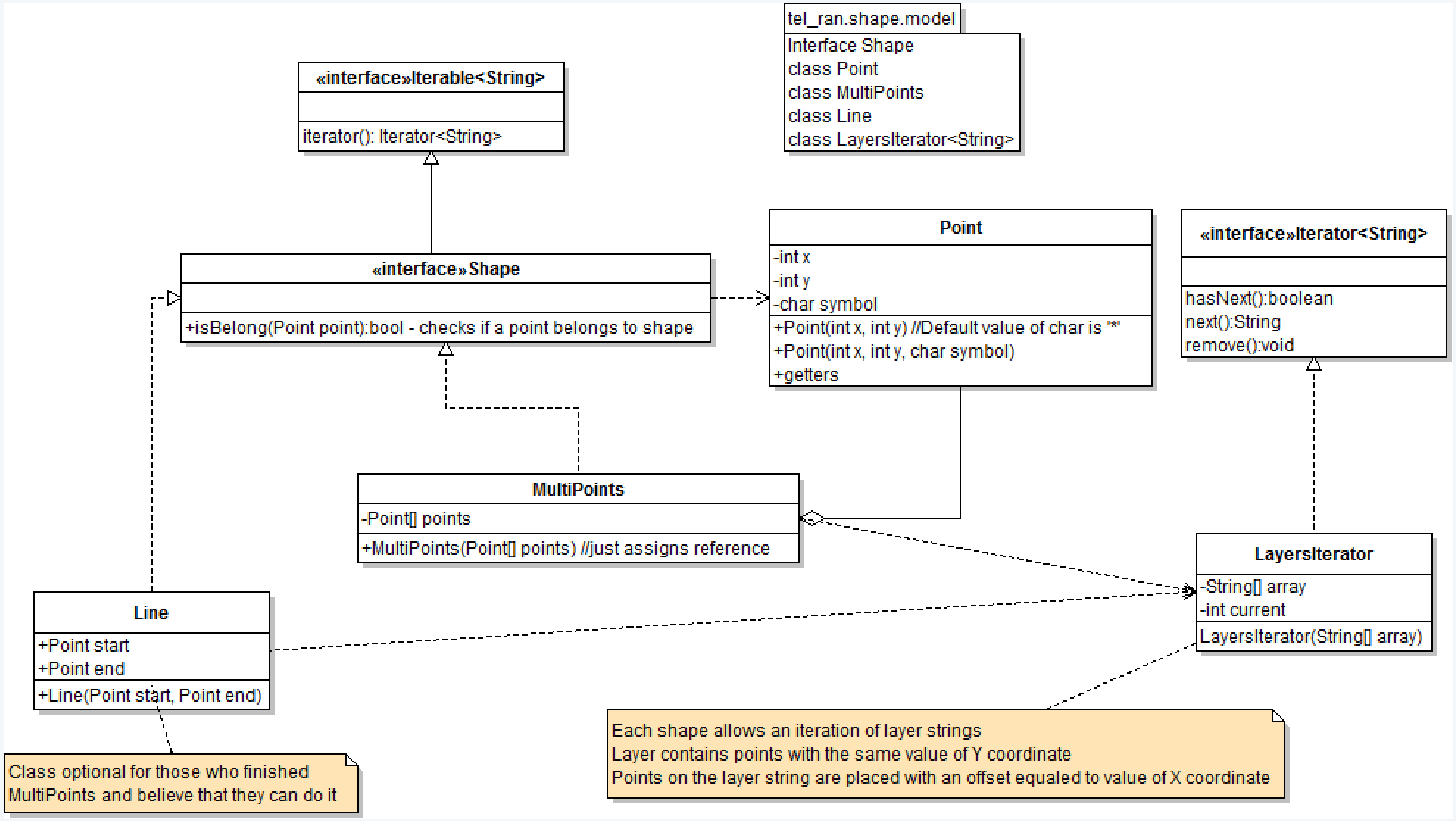
+ShopFunctionalTest(Shop shop, ShopView view, ProductItem[] item

Constructor fills all fields and creates shop
based on an array of product items
Method run prints all shop's product items,
prints all shop's items by name
prints the product item by bar code

Exercise-Shape Application

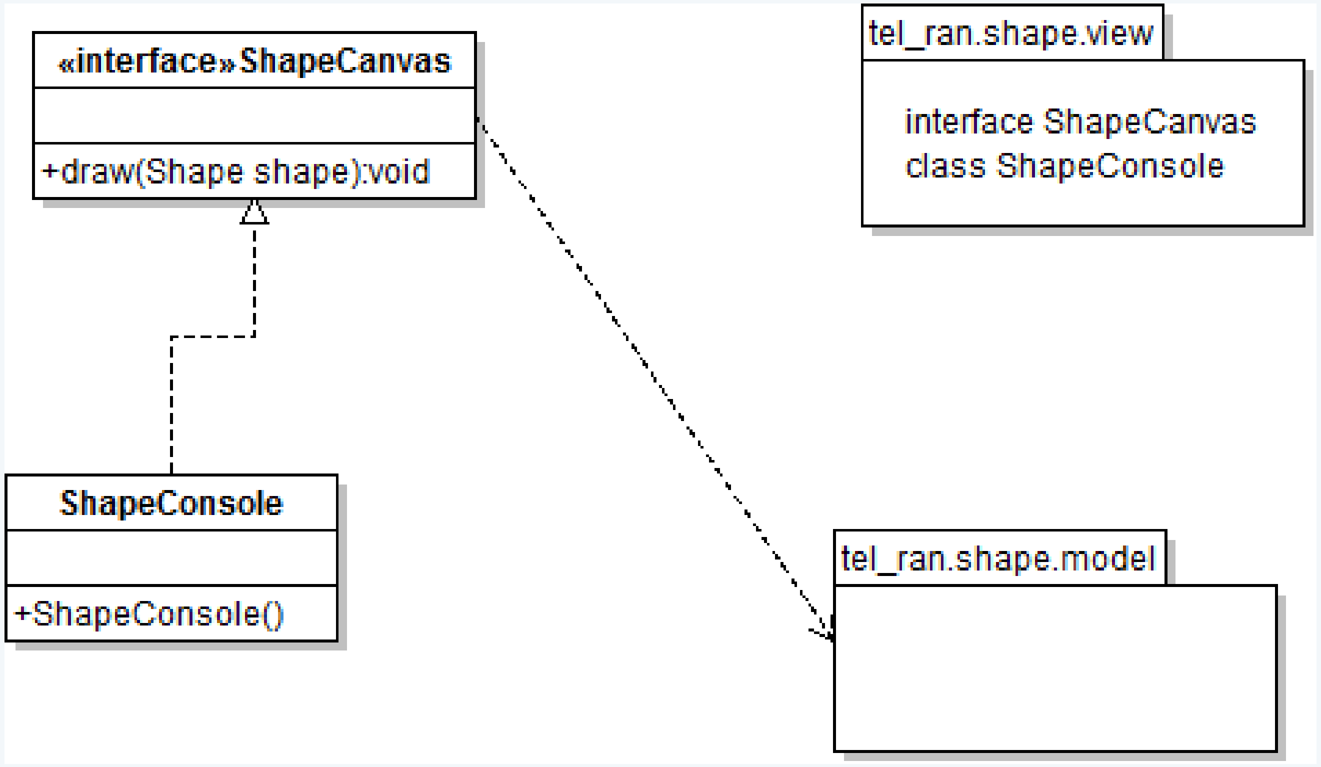
- Develop package tel_ran.shape.model
  - Class Line optional for those who have finished class MultiPoints
- Develop package tel_ran.shape.view
- Develop package tel_ran.shape.controller

Package tel_ran.shape.model

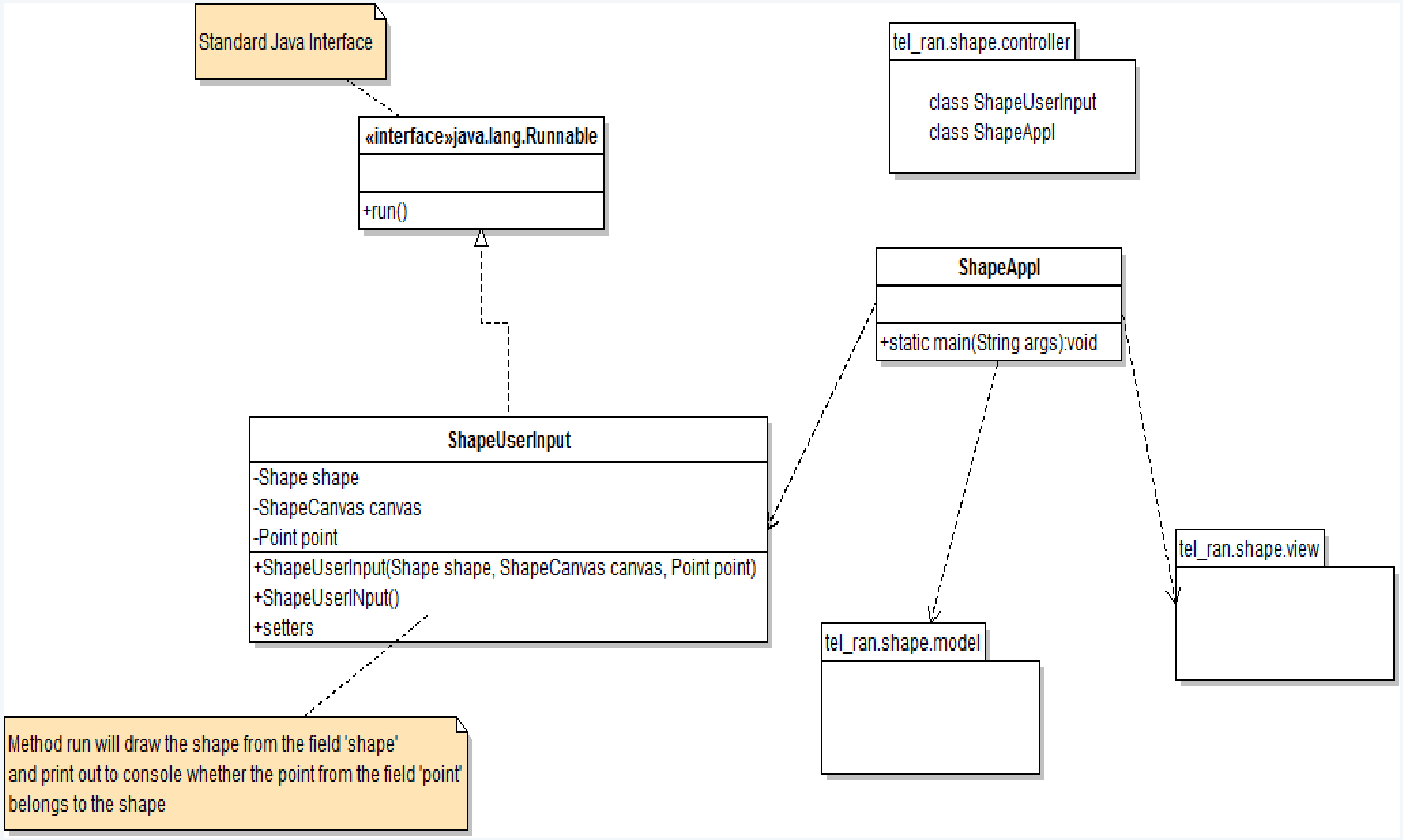


tel_ran.shape.model
Interface Shape
class Point
class MultiPoints
class Line
class LayersIterator<String>
«interface»Iterable<String>
iterator(): Iterator<String>
«interface»Shape
+isBelong(Point point):bool - checks if a point belongs to shape
Point
-int x
-int y
-char symbol
+Point(int x, int y) //Default value of char is '*'
+Point(int x, int y, char symbol)
+getters
«interface»Iterator<String>
hasNext():boolean
next():String
remove():void
MultiPoints
-Point[] points
+MultiPoints(Point[] points) //just assigns reference
LayersIterator
-String[] array
-int current
LayersIterator(String[] array)
Line
+Point start
+Point end
+Line(Point start, Point end)
Class optional for those who finished MultiPoints and believe that they can do it
Each shape allows an iteration of layer strings
Layer contains points with the same value of Y coordinate
Points on the layer string are placed with an offset equaled to value of X coordinate

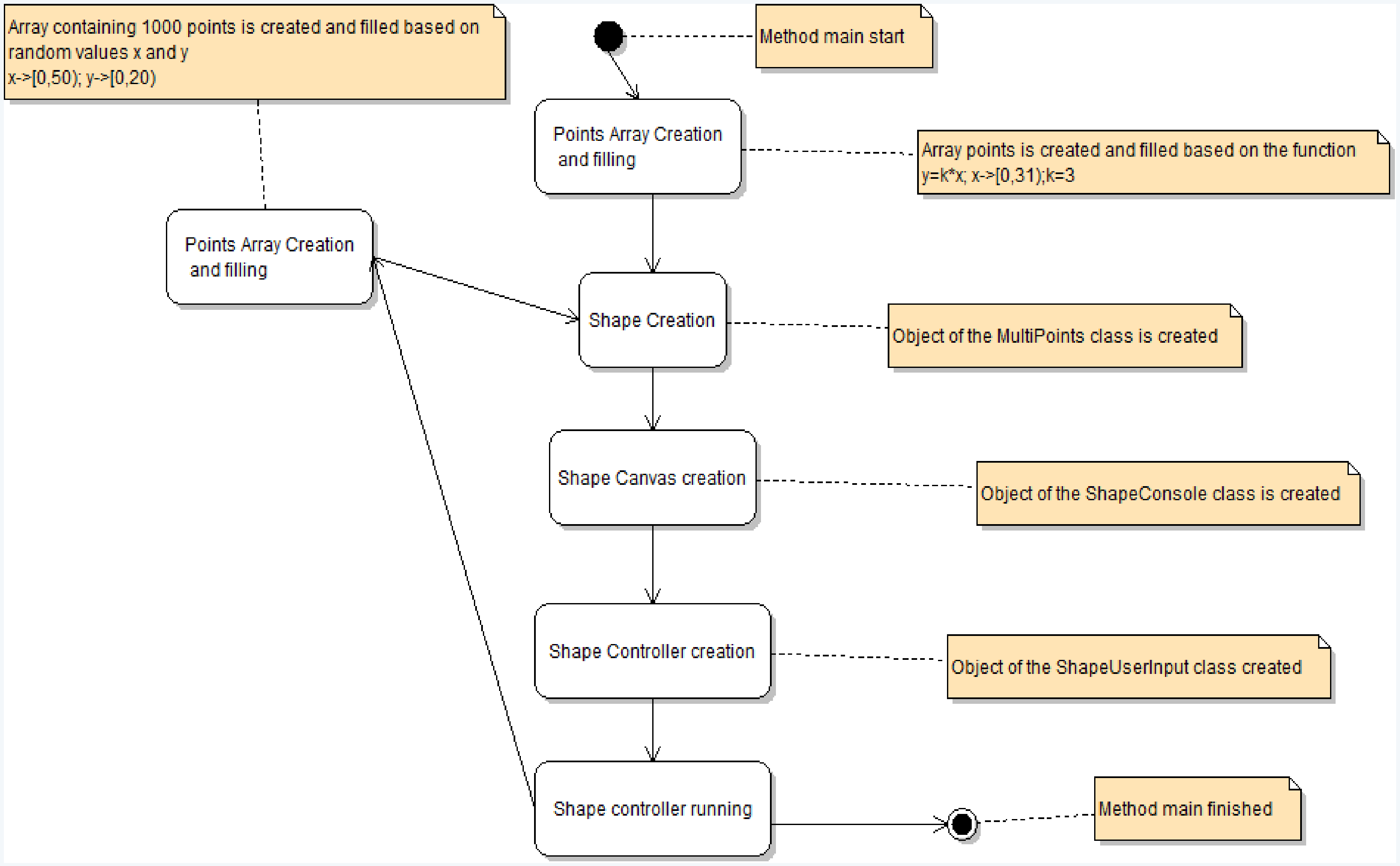
Package tel_ran.shape.view



Package tel_ran.shape.controller


Standard Java Interface
«interface»java.lang.Runnable
+run()
tel_ran.shape.controller
class ShapeUserInput
class ShapeAppl
ShapeAppl
+static main(String args):void
ShapeUserInput
-Shape shape
-ShapeCanvas canvas
-Point point
+ShapeUserInput(Shape shape, ShapeCanvas canvas, Point point)
+ShapeUserINput()
+setters
tel_ran.shape.view
tel_ran.shape.model
Method run will draw the shape from the field 'shape' and print out to console whether the point from the field 'point' belongs to the shape

Method main Activity Diagram


Array containing 1000 points is created and filled based on random values x and y x->[0,50); y->[0,20)
Method main start
Points Array Creation and filling
Array points is created and filled based on the function y=k*x; x->[0,31);k=3
Points Array Creation and filling
Shape Creation
Object of the MultiPoints class is created
Shape Canvas creation
Object of the ShapeConsole class is created
Shape Controller creation
Object of the ShapeUserInput class created
Shape controller running
Method main finished

Exercise Update of Shape Application to MVC

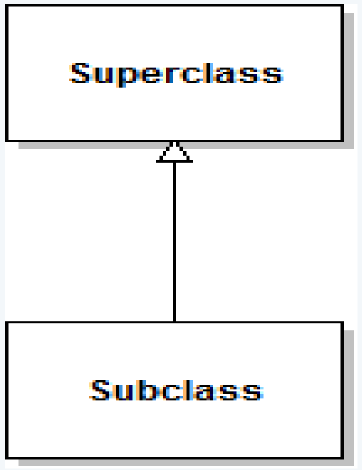
- Solution of Shape application with no separation between logics, presentation and controlling may be found in Dropbox at

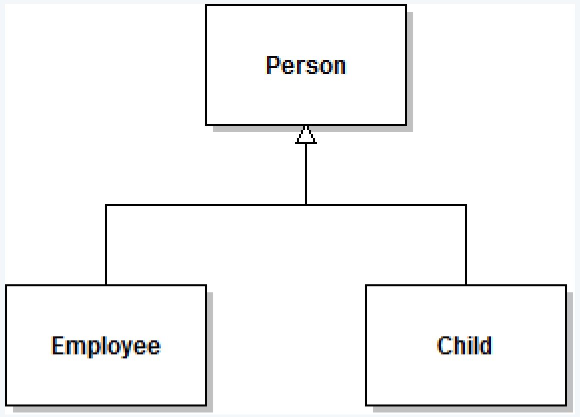
…/Solution/HW_ShapeNoMVC

  – This solution should be updated in the MVC form according to the UML model (61-63)

Java Inheritance

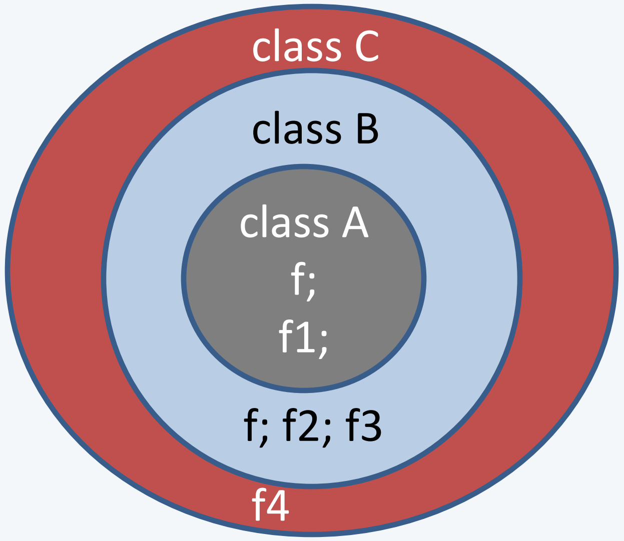
- Subclass receives all fields from Superclass that prevents copy/paste of the common data/functions.
  - For example there are three classes: ***person, employee, child***. All these classes have such common data as identity number, name, age, etc. Instead of copying code of the methods working with the common data we may use the inheritance feature.





Inheritance Terminology

- Class Contract - all class public methods through which the private class members may be accessible
- Class Protocol – all public methods the user may call through a class object
  - Protocol of a sub class equals the union of the sub class contract and protocol of a super class
- Protected class field – a field which may be accessible from the class's methods and from sub class's methods but inaccessible from outside. The better practice is to apply the protected methods instead of the protected data members.

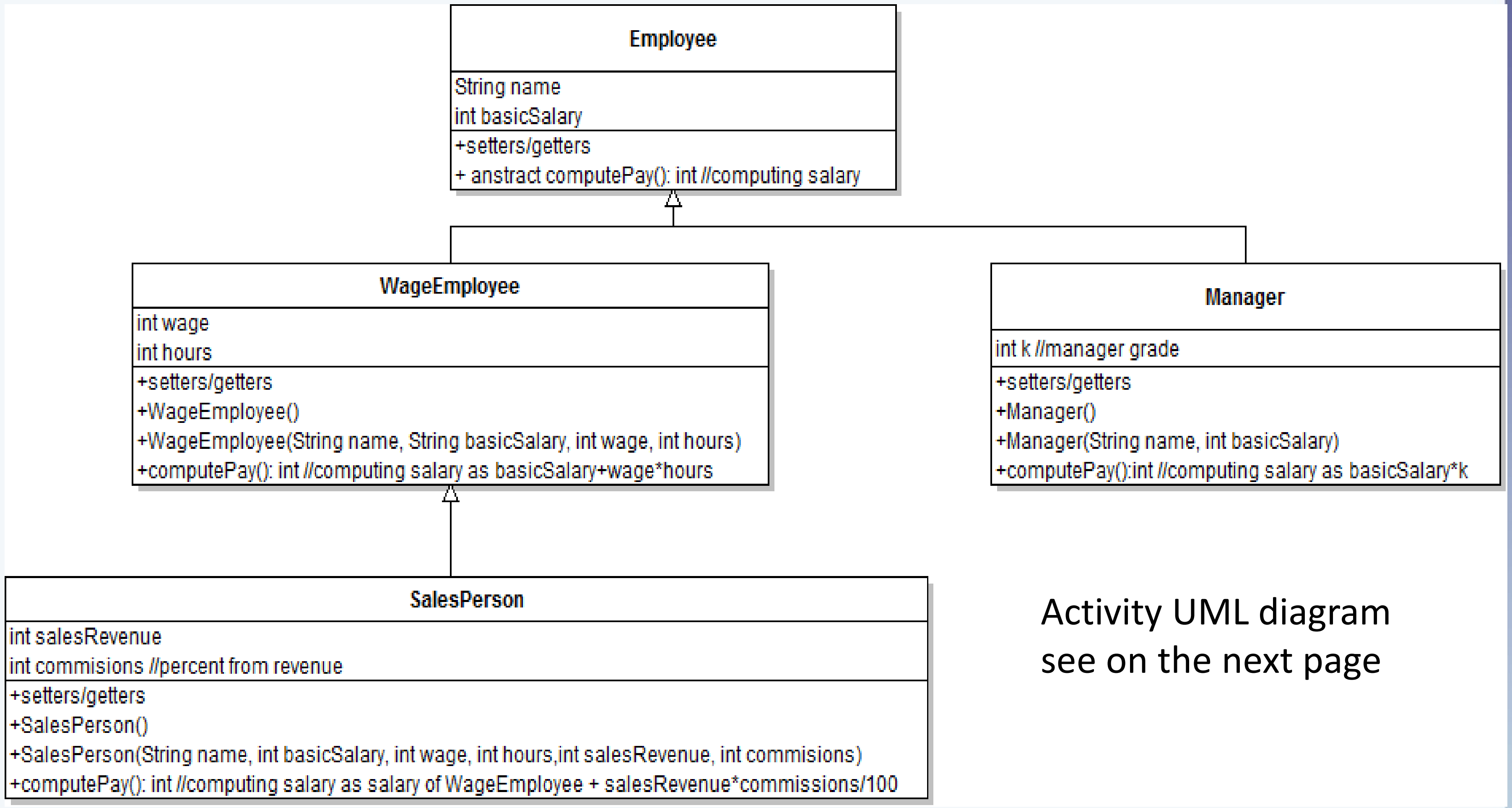


Contract-Protocol Example

Abstract Class

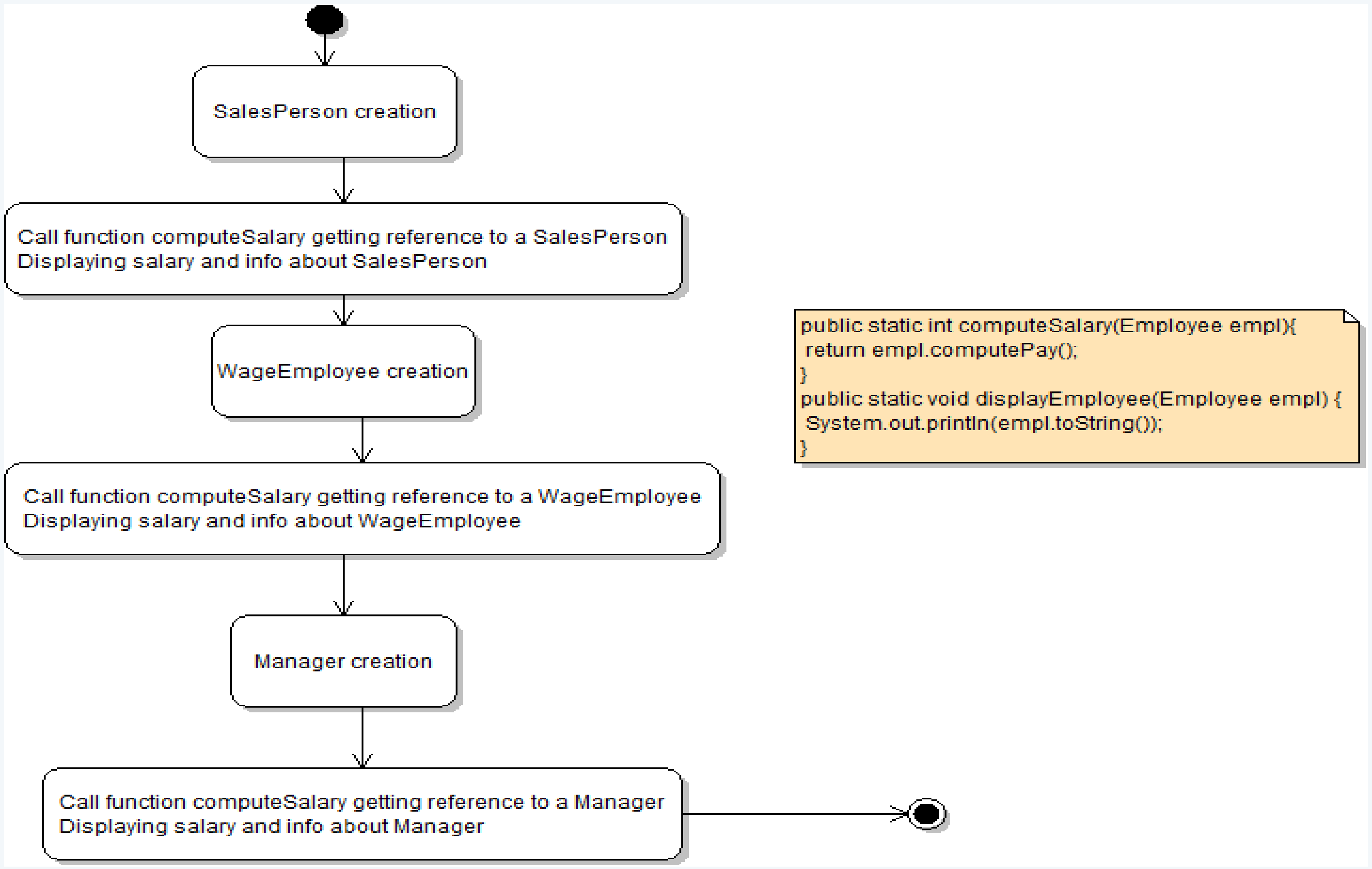
- Abstract class is a class with ***abstract*** keyword
- Abstract function is a function with ***abstract*** keyword
  - Abstract function contains just function prototype
- The class containing abstract function may be only the abstract class
- No object of an abstract class

Exercise- Develop UML Employee Inheritance


Employee
String name
int basicSalary
+setters/getters
+ anstract computePay(): int //computing salary
WageEmployee
int wage
int hours
+setters/getters
+WageEmployee()
+WageEmployee(String name, String basicSalary, int wage, int hours)
+computePay(): int //computing salary as basicSalary+wage*hours
Manager
int k //manager grade
+setters/getters
+Manager()
+Manager(String name, int basicSalary)
+computePay():int //computing salary as basicSalary*k
SalesPerson
int salesRevenue
int commisions //percent from revenue
+setters/getters
+SalesPerson()
+SalesPerson(String name, int basicSalary, int wage, int hours,int salesRevenue, int commisions)
+computePay(): int //computing salary as salary of WageEmployee + salesRevenue*commissions/100

Activity UML diagram see on the next page

Activity Diagram for Previous Exercise


SalesPerson creation
Call function computeSalary getting reference to a SalesPerson Displaying salary and info about SalesPerson
WageEmployee creation
Call function computeSalary getting reference to a WageEmployee Displaying salary and info about WageEmployee
Manager creation
Call function computeSalary getting reference to a Manager Displaying salary and info about Manager

```
public static int computeSalary(Employee empl){
 return empl.computePay();
}
public static void displayEmployee(Employee empl) {
 System.out.println(empl.toString());
}
```

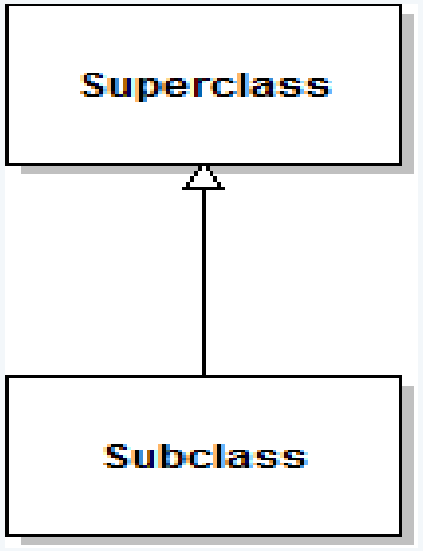
Class Object

- Non abstract class
- All classes inherit from class Object
  - All classes have methods from Object such as ***equals(), toString()***
- Allows writing universal methods
  - Examples:
    - method ***println(Object obj)*** may get an object of any class based on the method ***toString()***
    - Method equals of the class ***Arrays*** may equal any arrays based on the method ***equals()***

Up/Down Casts

- Up cast
  - From Subclass to Superclass
  - Implicit form as Subclass is a kind of Superclass
- Down cast
  - From Superclass to Subclass
  - Explicit form as Superclass is not a kind of Subclass
  - Operator ***instanceof*** for checking whether a reference of a Superclass type points to an object of a Subclass
    - Returns true if it does

<reference> instanceof <Subclass name>



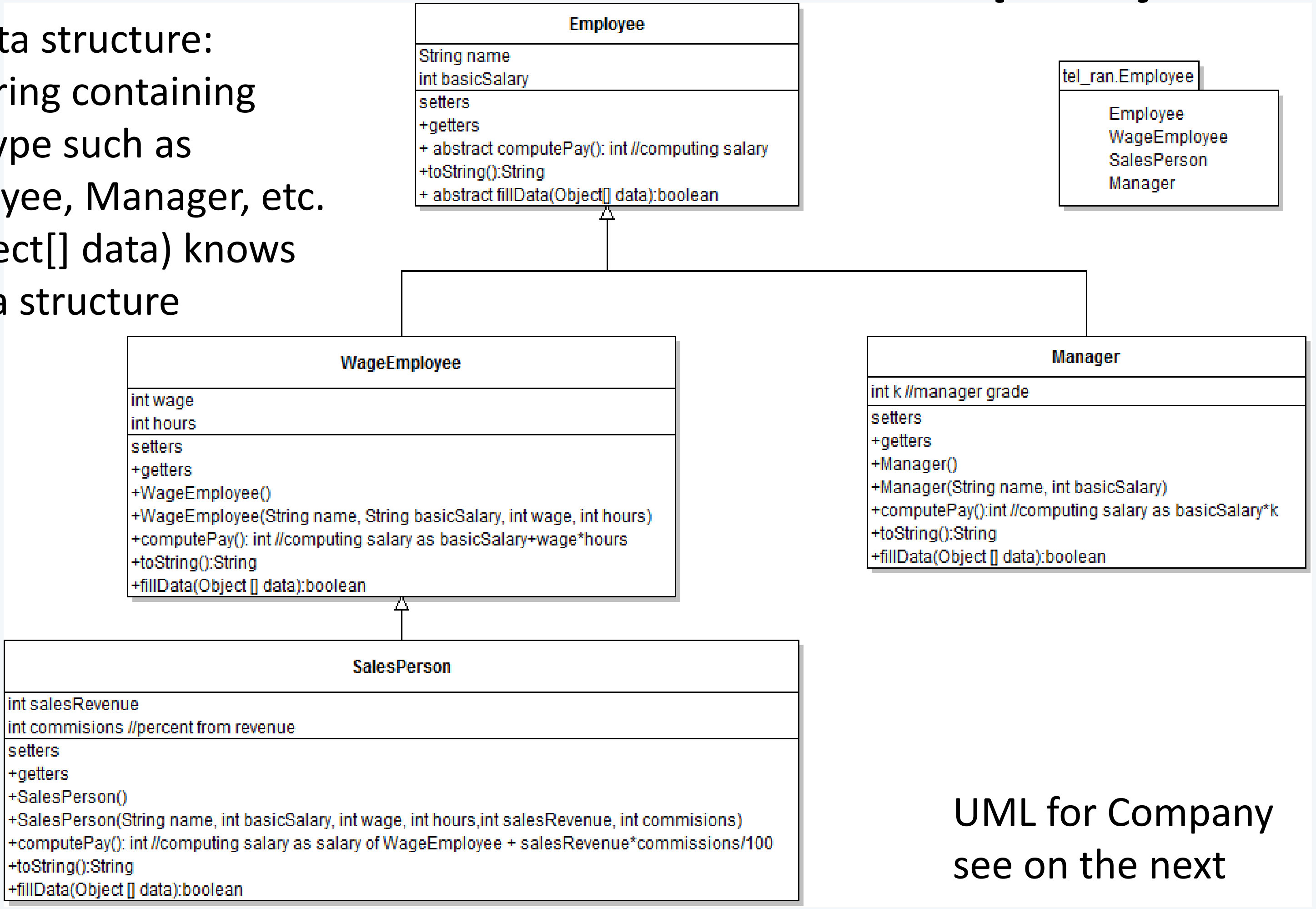
Class ***Class*** and Class Reflection

- Java has class naming as Class
- The main purpose of this class is to create object of the class the name of which may be received only at run time as string value. It is called as Reflection
  - Static Method ***forName(String typeName)*** allows the creation of an object of the class ***Class*** the name of which exists in the string ***typeName***
  - Method ***newInstance()*** of the class ***Class*** allows receiving instance of the class
  - Example assuming ***typeName*** contains the name of a class inheriting from ***Employee***

```
Class cl=Class.forName(typeName);
Employee empl=(Employee)cl.newInstance();
```

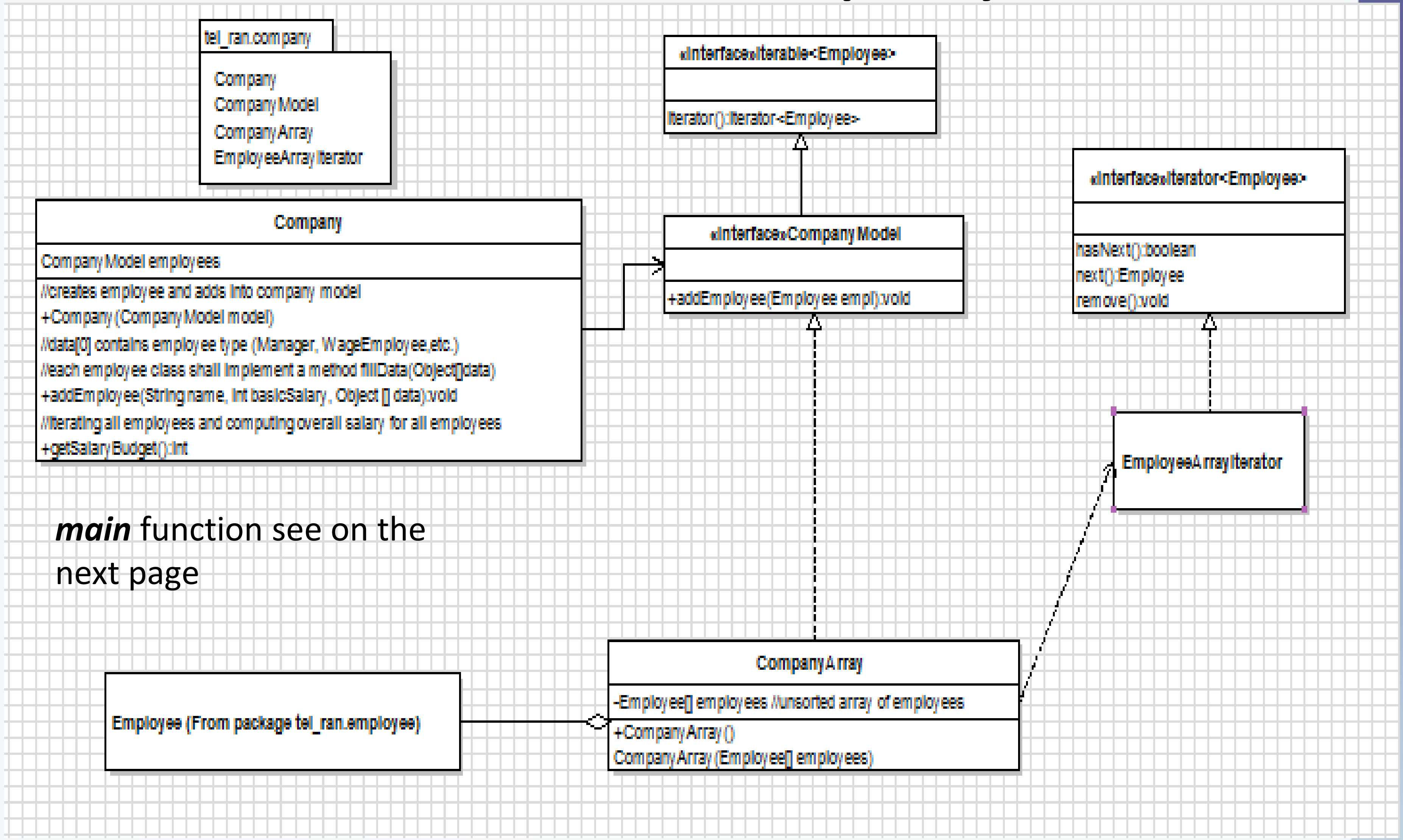
Exercise- Fill Data for Employee

Object [] data structure:
Data[0] – string containing employee type such as WageEmployee, Manager, etc.
fillData(Object[] data) knows specific data structure


Employee
String name
int basicSalary
setters
+getters
+ abstract computePay(): int //computing salary
+toString():String
+ abstract fillData(Object[] data):boolean
tel_ran.Employee
Employee
WageEmployee
SalesPerson
Manager
WageEmployee
int wage
int hours
setters
+getters
+WageEmployee()
+WageEmployee(String name, String basicSalary, int wage, int hours)
+computePay(): int //computing salary as basicSalary+wage*hours
+toString():String
+fillData(Object [] data):boolean
Manager
int k //manager grade
setters
+getters
+Manager()
+Manager(String name, int basicSalary)
+computePay():int //computing salary as basicSalary*k
+toString():String
+fillData(Object [] data):boolean
SalesPerson
int salesRevenue
int commisions //percent from revenue
setters
+getters
+SalesPerson()
+SalesPerson(String name, int basicSalary, int wage, int hours,int salesRevenue, int commisions)
+computePay(): int //computing salary as salary of WageEmployee + salesRevenue*commissions/100
+toString():String
+fillData(Object [] data):boolean

UML for Company see on the next page

Exercise-Company


tel_ran.company
Company
CompanyModel
CompanyArray
EmployeeArrayIterator
«interface»Iterable<Employee>
iterator():Iterator<Employee>
«interface»Iterator<Employee>
hasNext():boolean
next():Employee
remove():void
Company
CompanyModel employees
//creates employee and adds into company model
+Company(CompanyModel model)
//data[0] contains employee type (Manager, WageEmployee,etc.)
//each employee class shall implement a method fillData(Object[]data)
+addEmployee(String name, int basicSalary, Object [] data):void
//iterating all employees and computing overall salary for all employees
+getSalaryBudget():int
«interface»CompanyModel
+addEmployee(Employee empl):void
EmployeeArrayIterator
Employee (From package tel_ran.employee)
CompanyArray
-Employee[] employees //unsorted array of employees
+CompanyArray()
CompanyArray(Employee[] employees)

main function see on the next page

Exercise Company - ***main*** Function

```
public static void main(String[] args) {
            Object [] managerData=new Object[2];
            managerData[0]="Manager";
            managerData[1]=15;
            Object [] wageEmplData=new Object[3];
            wageEmplData[0]="WageEmployee";
            wageEmplData[1]=100;
            wageEmplData[2]=150;
            Object[] salesPersonData=new Object[5];
            salesPersonData[0]="SalesPerson";
            salesPersonData[1]=50;
            salesPersonData[2]=100;
            salesPersonData[3]=10000;
            salesPersonData[4]=2;
            CompanyModel model=new CompanyArray();
            Company company=new Company(model);
            company.addEmployee("Moshe",1000,managerData);
            company.addEmployee("Serg", 1000, wageEmplData);
            company.addEmployee("Vova",500,salesPersonData);
            System.out.println(company.getSalaryBudget());
}
```

Final Keyword

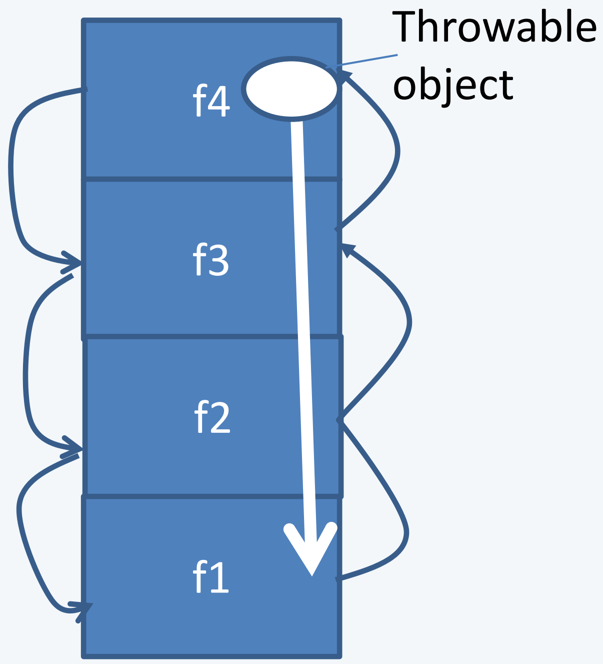
- Final class
  - Couldn’t be Superclass as it is impossible to inherit from a final class
- Final method
  - Couldn’t be overridden in a Subclass
- Final data member
  - Couldn’t be changed

Interface vs. Abstract class

- Interface
  - Purpose
    - To write universal methods independent on the implementation details
  - Either static final data member or a function prototype
  - A class may implement as many as needed interfaces
  - Implementation class should override all the interface function otherwise it must be an abstract class
- Abstract class
  - Purpose
    - To be just super class for writing universal methods independent on the implementation details and for inheritance of common methods and data
  - May contain both abstract and real methods, and data members
  - A class may inherit just from one abstract class
  - In the case a class has at least one abstract method it must be the abstract one

Exceptions

- Regular stream of methods
  - Function caller (i.e. f1()) calls function callee (i.e. f2()) and waits for finishing of callee at the call point
- ***throw –*** operator throws ***Throwable*** object which rewinds the stack and changes the regular stream of methods
- ***Throwable*** object may be caught and processed



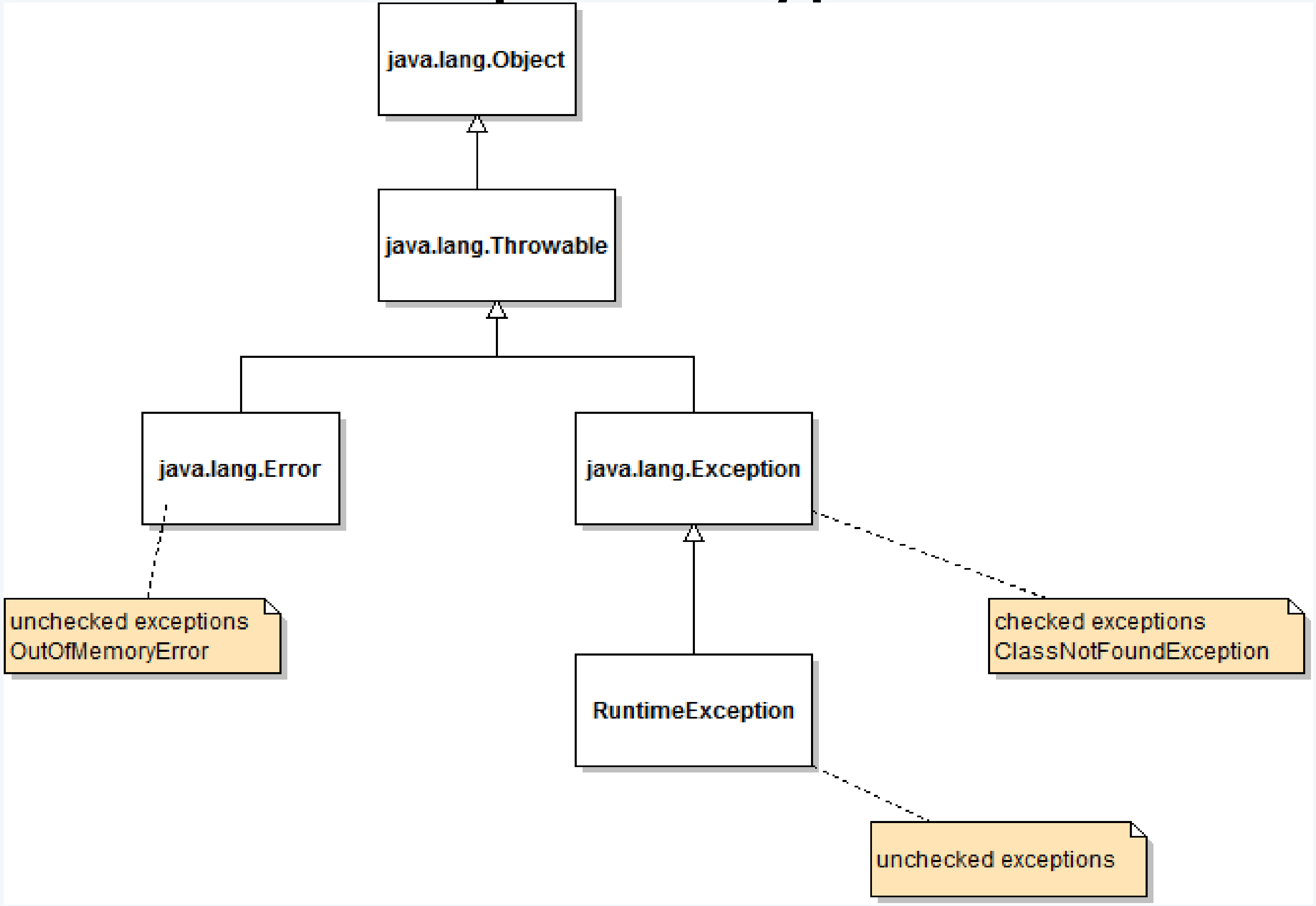
Throwable object may rewind the stack changing reqular order of the methods

How to Catch Exceptions

- Catch exceptions using ***try*** blocks:

```
try {
// statements that might cause exceptions
// possibly including function calls
}
catch ( <exception-1> <id-1> ) {
// statements to handle this exception
} catch ( <exception-2> <id-2> ) {
// statements to handle this exception . . .
} finally {
// statements to execute every time this try block executes
}
```

Exception Types



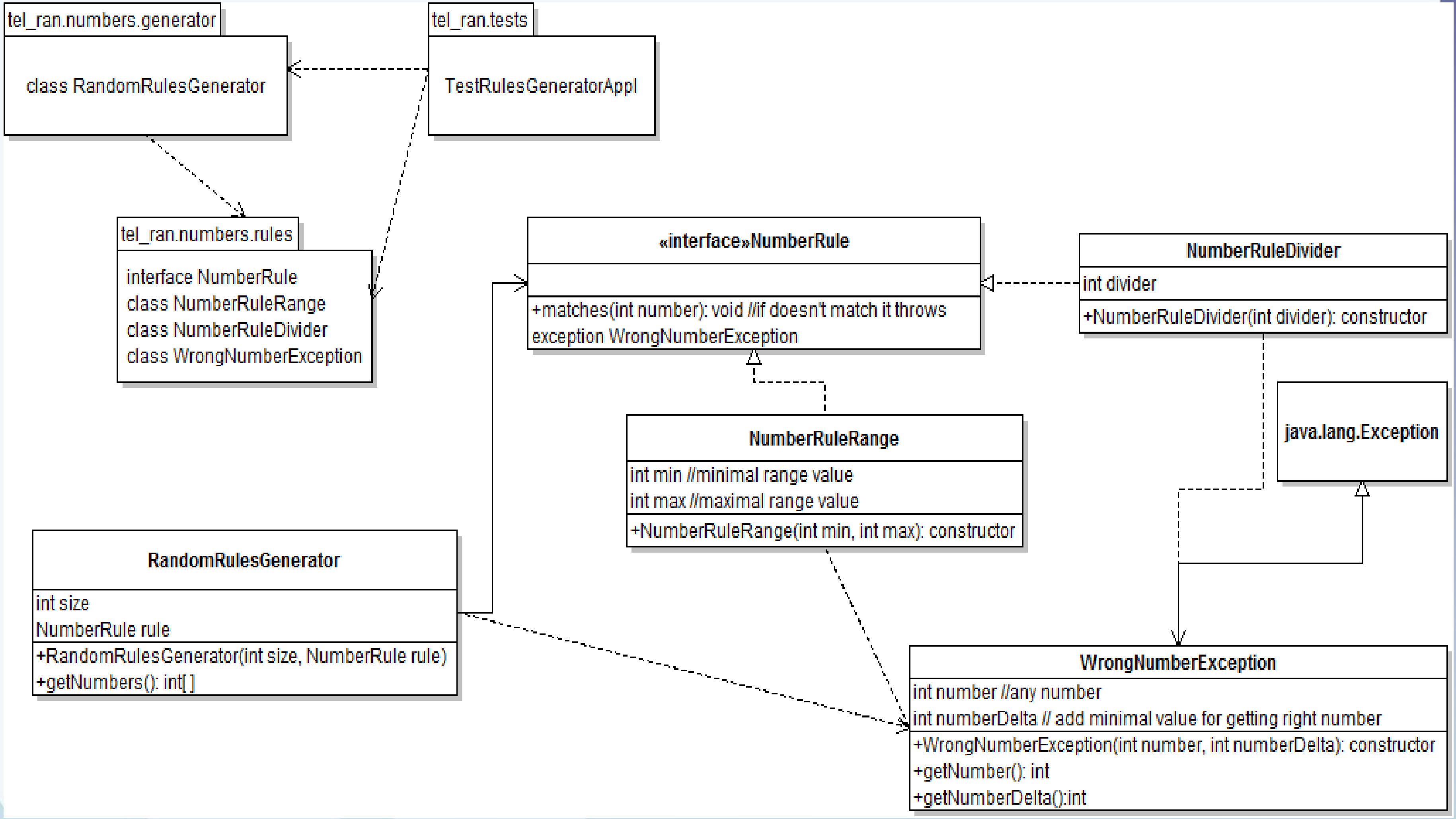
Checked and Unchecked Exceptions

- Every exception is either a **checked** exception or an **unchecked** exception. If a method includes code that could cause a **checked** exception to be thrown, then:
  - the exception must be declared in the method header, using a **throws** clause, or
  - the code that might cause the exception to be thrown must be inside a *try* block with a *catch* clause for that exception.

Exception class

- Most widespread superclass for user created exceptions to be thrown and caught
- Most widespread methods are
  - Constructor: Exception(String message)
  - getMessage():String
  - printStackTrace():void
- Even if no a need to add new data and/or methods it is acceptable to create a user specific class inheriting from ***Exception***

Exercise – Numbers Generator

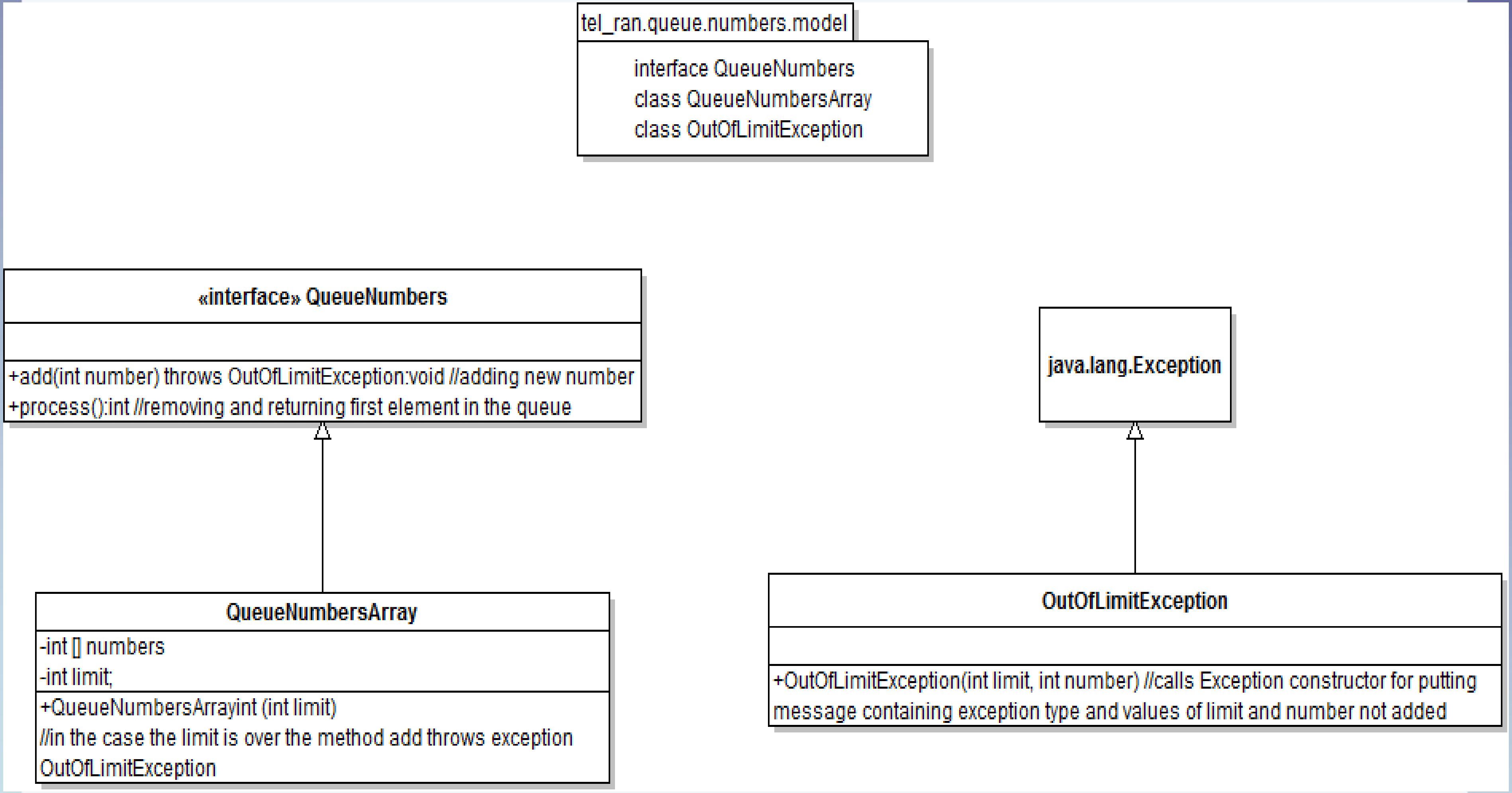

tel_ran.numbers.generator
class RandomRulesGenerator
tel_ran.tests
TestRulesGeneratorAppl
tel_ran.numbers.rules
interface NumberRule
class NumberRuleRange
class NumberRuleDivider
class WrongNumberException
«interface»NumberRule
+matches(int number): void //if doesn't match it throws exception WrongNumberException
NumberRuleDivider
int divider
+NumberRuleDivider(int divider): constructor
java.lang.Exception
NumberRuleRange
int min //minimal range value
int max //maximal range value
+NumberRuleRange(int min, int max): constructor
RandomRulesGenerator
int size
NumberRule rule
+RandomRulesGenerator(int size, NumberRule rule)
+getNumbers(): int[ ]
WrongNumberException
int number //any number
int numberDelta // add minimal value for getting right number
+WrongNumberException(int number, int numberDelta): constructor
+getNumber(): int
+getNumberDelta():int

Exercise - QueueNumbers

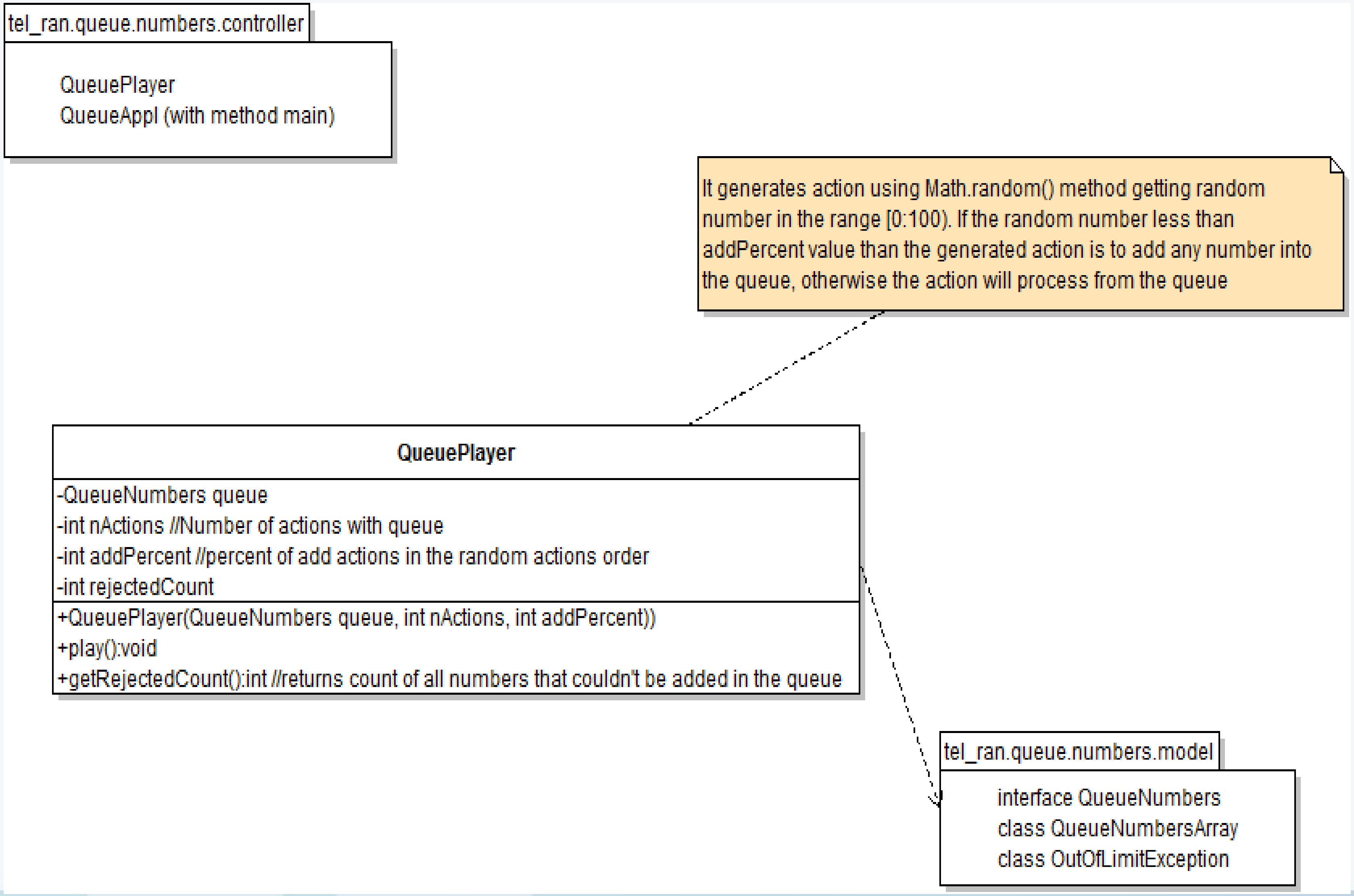
- Develop queue numbers model according to the UML diagram at page 85
- Develop queue numbers controller according to the UML diagram at page 86
- Write application class with the following method main

```
 static final int LIMIT=100;
 static  final int ADD_PROBABILITY=80;
Static final int N_ACTIONS = 10000;
 public static void main(String args[]){
		QueueNumbers qn=new QueueNumbersArray(LIMIT);
QueuePlayer qp=new QueuePlayer(qn,N_ACTIONS,ADD_PROBABILITY) ;
qp.play();
System.out.println(qp.getRejectedCount ());
}
```

UML Diagram – Queue Numbers Model


tel_ran.queue.numbers.model
interface QueueNumbers
class QueueNumbersArray
class OutOfLimitException
«interface» QueueNumbers
+add(int number) throws OutOfLimitException:void //adding new number
+process():int //removing and returning first element in the queue
java.lang.Exception
QueueNumbersArray
-int [] numbers
-int limit;
+QueueNumbersArrayint (int limit)
//in the case the limit is over the method add throws exception OutOfLimitException
OutOfLimitException
+OutOfLimitException(int limit, int number) //calls Exception constructor for putting message containing exception type and values of limit and number not added

UML Diagram-Queue Controller



tel_ran.queue.numbers.controller

QueuePlayer
QueueAppl (with method main)

It generates action using Math.random() method getting random number in the range [0:100). If the random number less than addPercent value than the generated action is to add any number into the queue, otherwise the action will process from the queue

QueuePlayer

-QueueNumbers queue
-int nActions //Number of actions with queue
-int addPercent //percent of add actions in the random actions order
-int rejectedCount

+QueuePlayer(QueueNumbers queue, int nActions, int addPercent))
+play():void
+getRejectedCount():int //returns count of all numbers that couldn't be added in the queue

tel_ran.queue.numbers.model

interface QueueNumbers
class QueueNumbersArray
class OutOfLimitException

Exercise – Public Access the Immutable Data Member

- Write class ***PublicDM*** allowing public access to immutable data member of type ***int*** with no getter (like data member ***length*** in an array)
  - The value of this data member shall be initialized during constructing an object of the class ***PublicDM***
- Write class ***PublicDmTest*** containing the following ***main*** method

```
public static void main(String [] args) {
        PublicDM pdm7=new PublicDM(7);
        System.out.println(pdm7.imfield); //printed 7
        PublicDM pdm10=new PublicDM(10);
        System.out.println(pdm10.imfield); //printed 10


}
```

What is the problem in the below code?

```
class X {
public int final x=10;
private int a;
public X(int a) {this.a=a;}
public int getA() {return a;}
}
```

Exercise – Queuing System Application

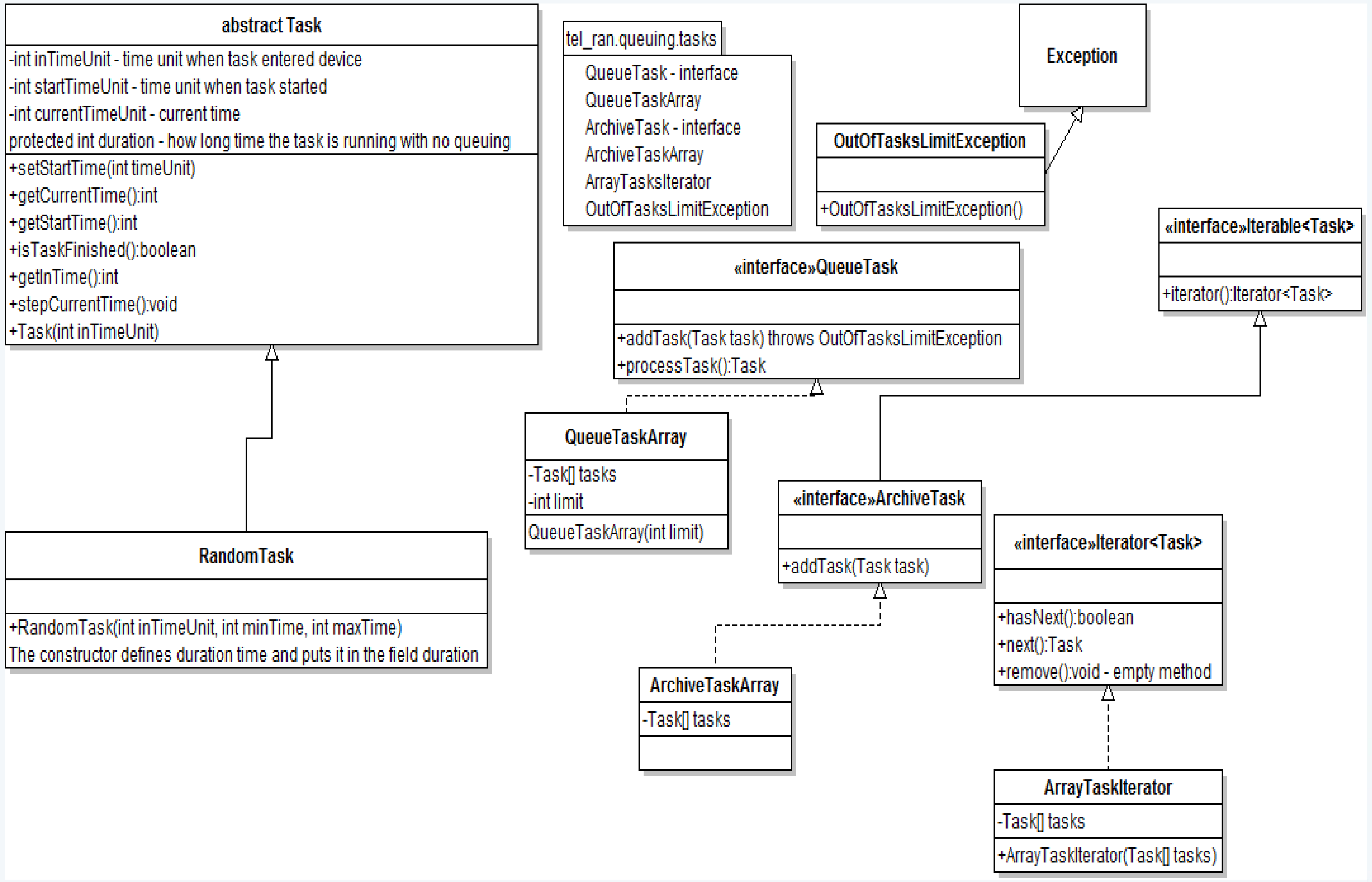
- Class QueuingSystemTest with method main on page 90
- Develop UML Class Diagram (Queuing System Tasks) on page 91
- Activity Diagram for Generator methods on pages 92
- Activity Diagram for QueuingDevice methods on page 93, 94
- Develop UML Class Diagram (Queuing System Elements) on page 95

Method ***main*** of Class QueuingSystemTest

```
pubic static void main(String args[]){
   QueueTask qt=new QueueTaskArray(QUEUE_LIMIT);
   ArchiveTask at=new ArchiveTaskArray();
  QueuingDevice device=new SingleChannelDevice(qt, at);
  Generator generator=new RandomGenerator(MIN_TASK_TIME, MAX_TASK_TIME);
              for(int i=0; i<N_ITERATIONS; i++){
                   Task task;
                    task=generator.getTask(i);
                    if(task != null)
                        device.startTask(task);
                    device.stepTimeUnit();
              }
    System.out.println("number of rejected tasks is "+device.getRejectedCount() );
    System.out.println("average waiting time is "+device.getAverageWaitingTime());
```

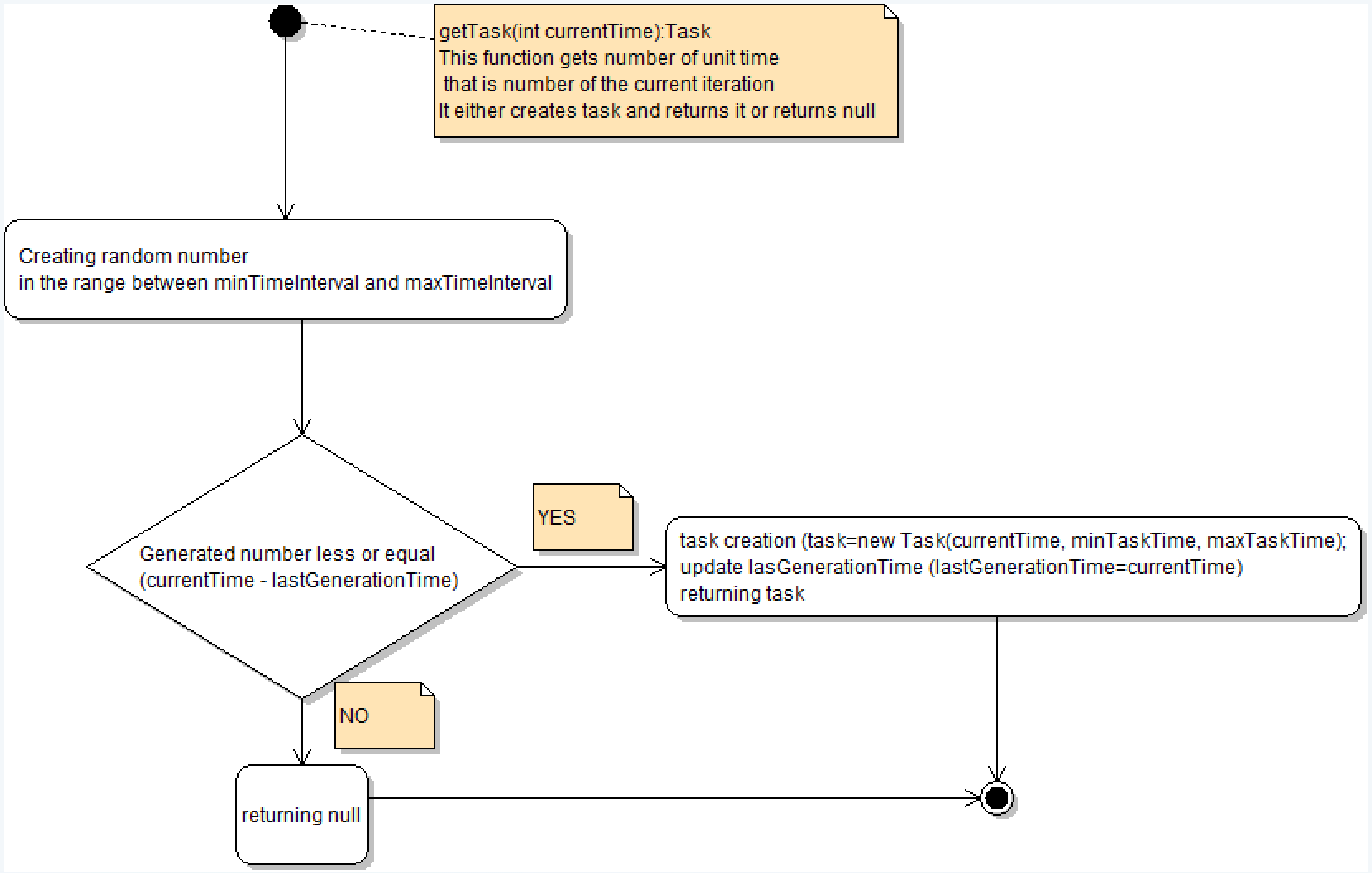
Simulation of a queuing system N_ITEARATIONS defines number of the time units. One iteration implies one unit time

UML Class Diagram – Queuing System Tasks

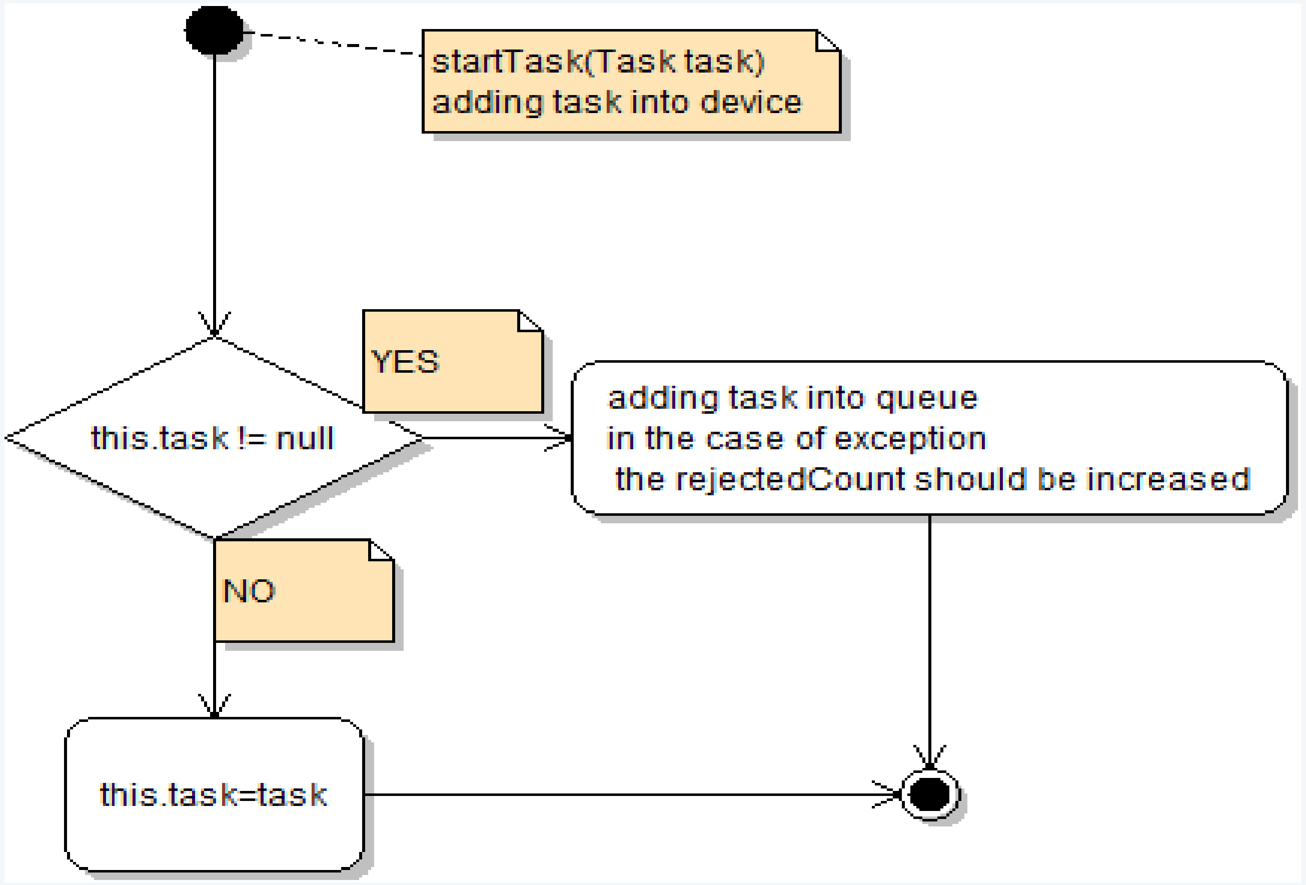


abstract Task
-int inTimeUnit - time unit when task entered device
-int startTimeUnit - time unit when task started
-int currentTimeUnit - current time
protected int duration - how long time the task is running with no queuing
+setStartTime(int timeUnit)
+getCurrentTime():int
+getStartTime():int
+isTaskFinished():boolean
+getInTime():int
+stepCurrentTime():void
+Task(int inTimeUnit)
tel_ran.queuing.tasks
QueueTask - interface
QueueTaskArray
ArchiveTask - interface
ArchiveTaskArray
ArrayTasksIterator
OutOfTasksLimitException
Exception
OutOfTasksLimitException
+OutOfTasksLimitException()
«interface»Iterable<Task>
+iterator():Iterator<Task>
«interface»QueueTask
+addTask(Task task) throws OutOfTasksLimitException
+processTask():Task
QueueTaskArray
-Task[] tasks
-int limit
QueueTaskArray(int limit)
«interface»ArchiveTask
+addTask(Task task)
«interface»Iterator<Task>
+hasNext():boolean
+next():Task
+remove():void - empty method
RandomTask
+RandomTask(int inTimeUnit, int minTime, int maxTime)
The constructor defines duration time and puts it in the field duration
ArchiveTaskArray
-Task[] tasks
ArrayTaskIterator
-Task[] tasks
+ArrayTaskIterator(Task[] tasks)

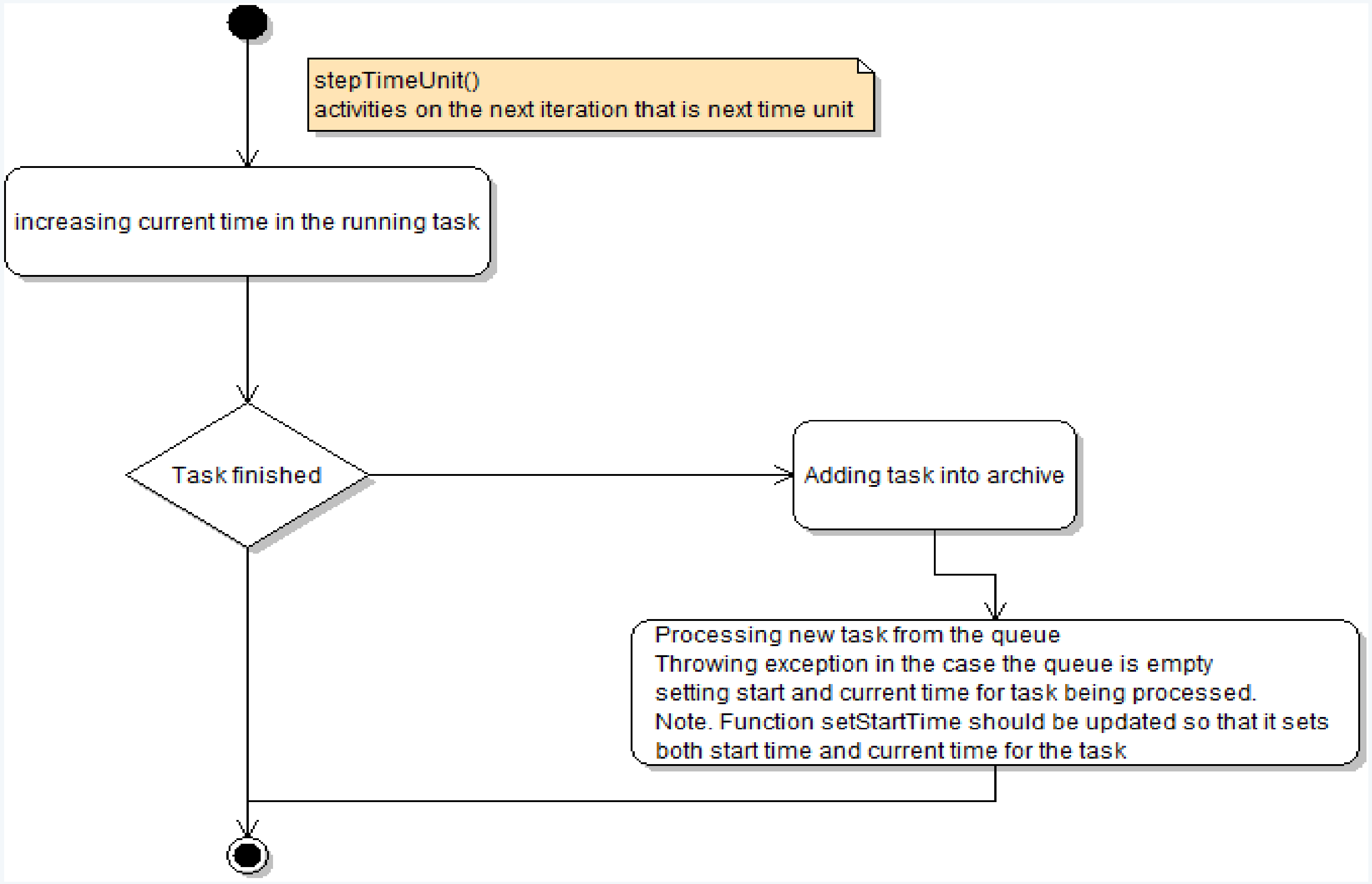
Activity Diagram - Generator


getTask(int currentTime):Task
This function gets number of unit time
that is number of the current iteration
It either creates task and returns it or returns null
Creating random number
in the range between minTimeInterval and maxTimeInterval
Generated number less or equal
(currentTime - lastGenerationTime)
YES
task creation (task=new Task(currentTime, minTaskTime, maxTaskTime);
update lasGenerationTime (lastGenerationTime=currentTime)
returning task
NO
returning null

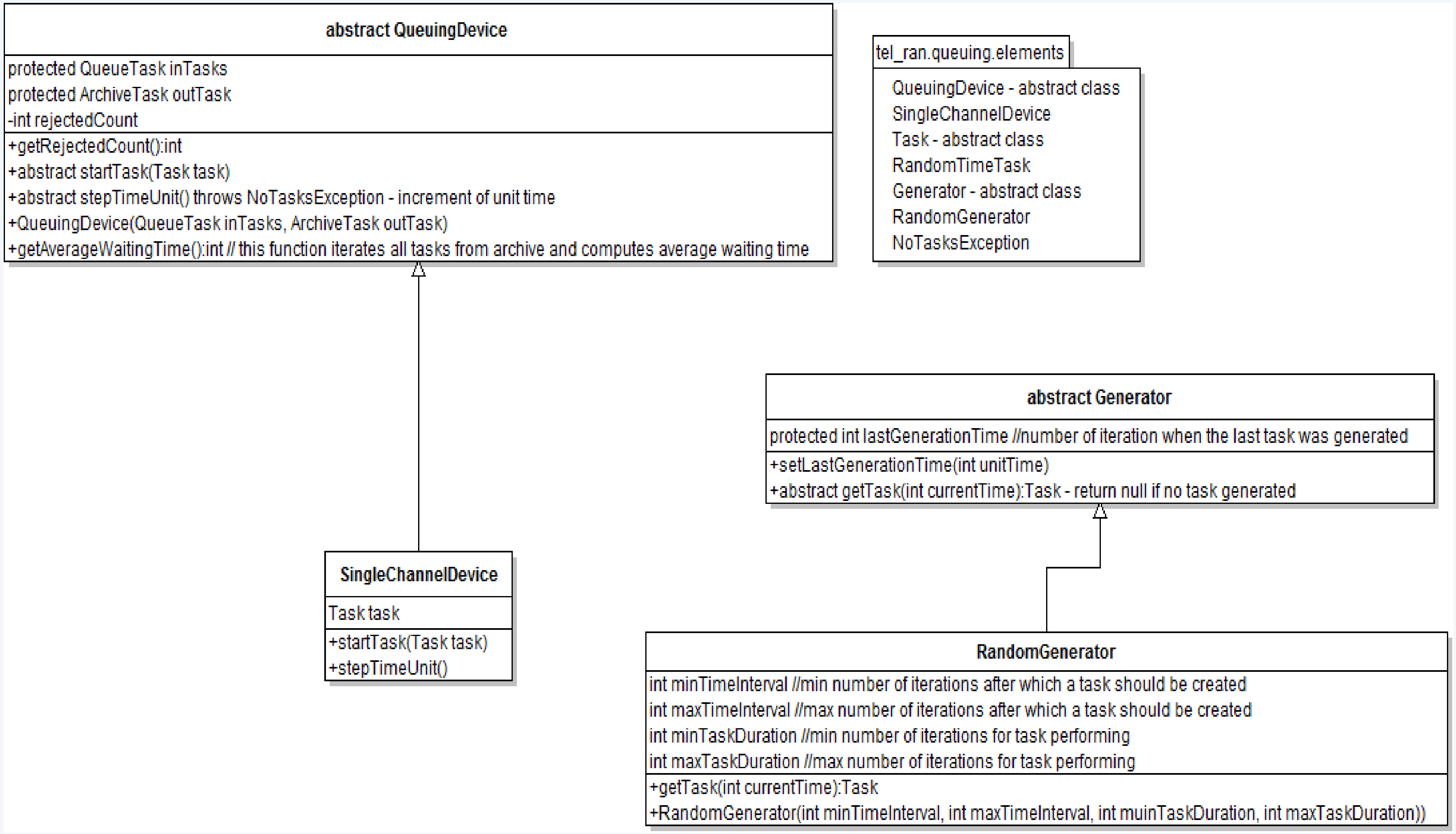
Activity Diagram – Queuing Device Start Task



Activity Diagram – Queuing Device Step Time Unit


stepTimeUnit()
activities on the next iteration that is next time unit
increasing current time in the running task
Task finished
Adding task into archive
Processing new task from the queue
Throwing exception in the case the queue is empty
setting start and current time for task being processed.
Note. Function setStartTime should be updated so that it sets both start time and current time for the task

UML Diagram – Queuing System Elements



protected QueueTask inTasks
protected ArchiveTask outTask
-int rejectedCount

+getRejectedCount():int
+abstract startTask(Task task)
+abstract stepTimeUnit() throws NoTasksException - increment of unit time
+QueuingDevice(QueueTask inTasks, ArchiveTask outTask)
+getAverageWaitingTime():int // this function iterates all tasks from archive and computes average waiting time

QueuingDevice - abstract class
SingleChannelDevice
Task - abstract class
RandomTimeTask
Generator - abstract class
RandomGenerator
NoTasksException

protected int lastGenerationTime //number of iteration when the last task was generated

+setLastGenerationTime(int unitTime)
+abstract getTask(int currentTime):Task - return null if no task generated

Task task

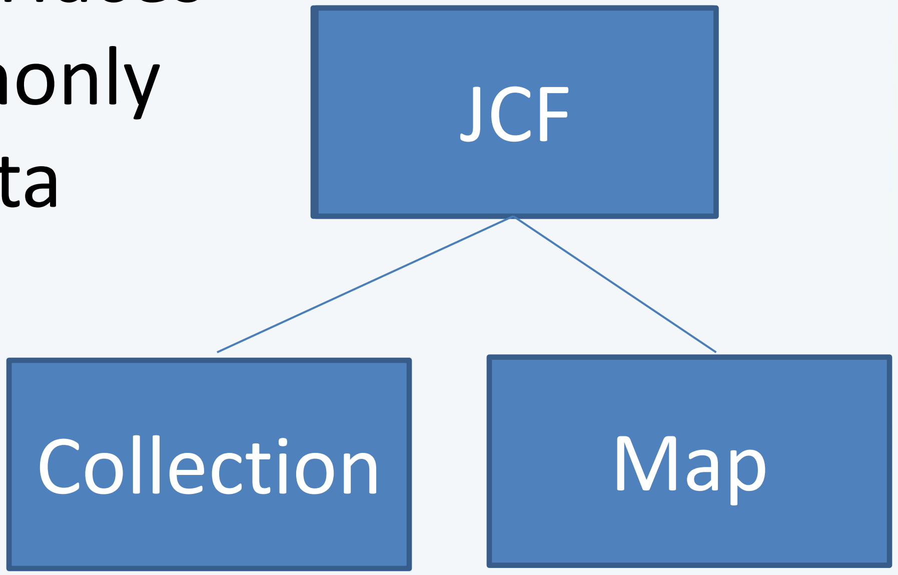
+startTask(Task task)
+stepTimeUnit()

int minTimeInterval //min number of iterations after which a task should be created
int maxTimeInterval //max number of iterations after which a task should be created
int minTaskDuration //min number of iterations for task performing
int maxTaskDuration //max number of iterations for task performing

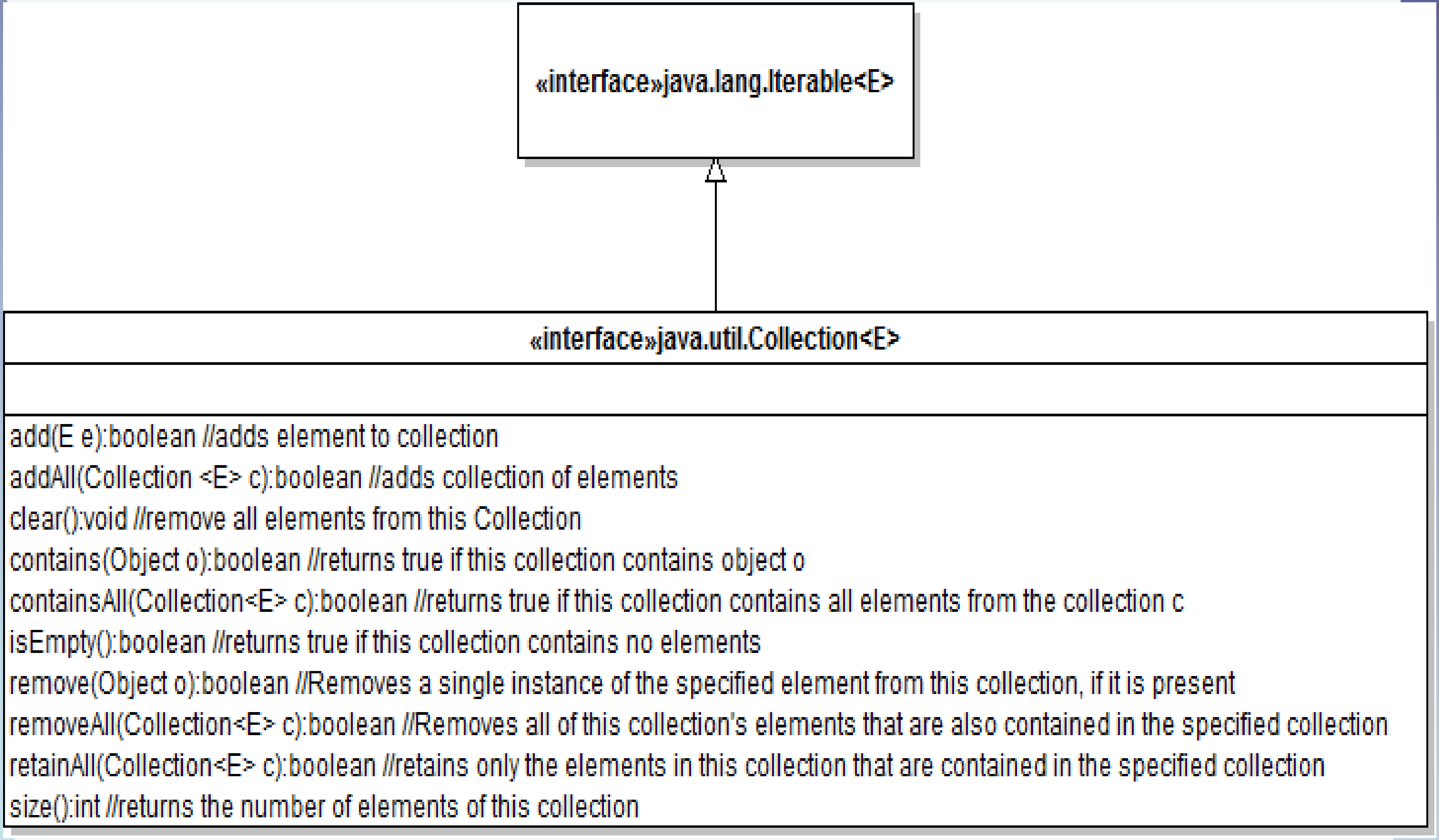
+getTask(int currentTime):Task
+RandomGenerator(int minTimeInterval, int maxTimeInterval, int muinTaskDuration, int maxTaskDuration))

Java Collections Framework (JCF) Collection

- Set of classes and interfaces that implement commonly reusable collection data structures
- Collection Interface
- Map Interface

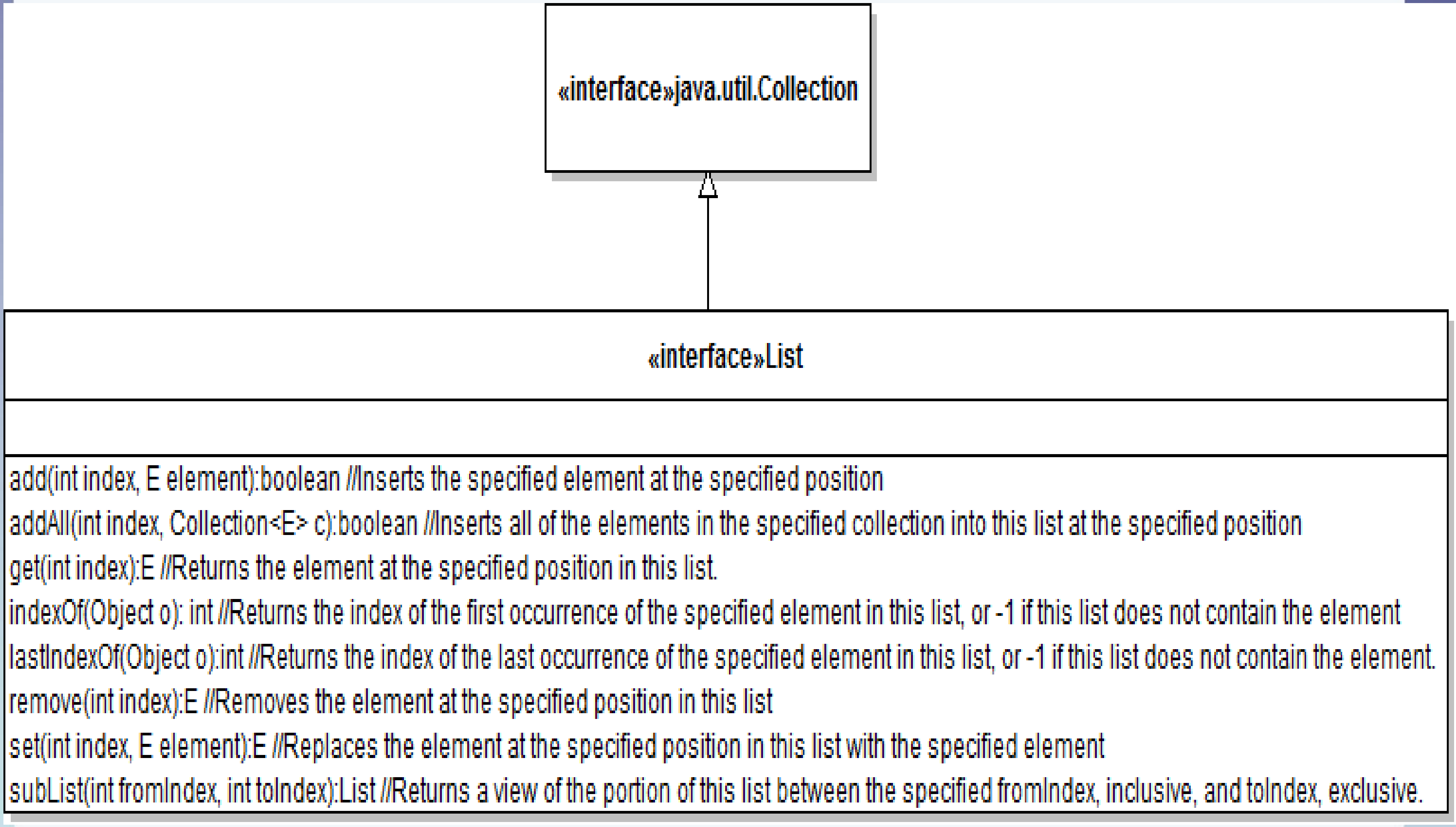


Collection Interface


«interface»java.lang.Iterable<E>
«interface»java.util.Collection<E>

add(E e):boolean //adds element to collection
addAll(Collection <E> c):boolean //adds collection of elements
clear():void //remove all elements from this Collection
contains(Object o):boolean //returns true if this collection contains object o
containsAll(Collection<E> c):boolean //returns true if this collection contains all elements from the collection c
isEmpty():boolean //returns true if this collection contains no elements
remove(Object o):boolean //Removes a single instance of the specified element from this collection, if it is present
removeAll(Collection<E> c):boolean //Removes all of this collection's elements that are also contained in the specified collection
retainAll(Collection<E> c):boolean //retains only the elements in this collection that are contained in the specified collection
size():int //returns the number of elements of this collection

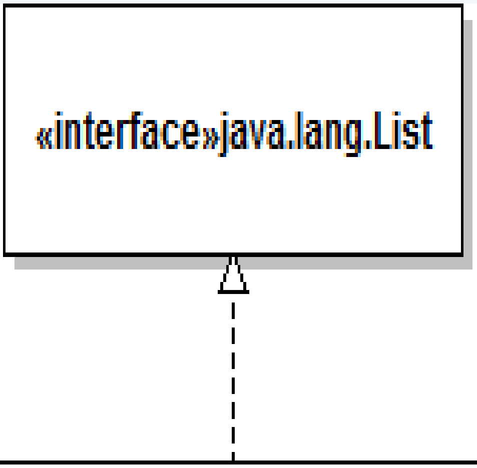
List Interface



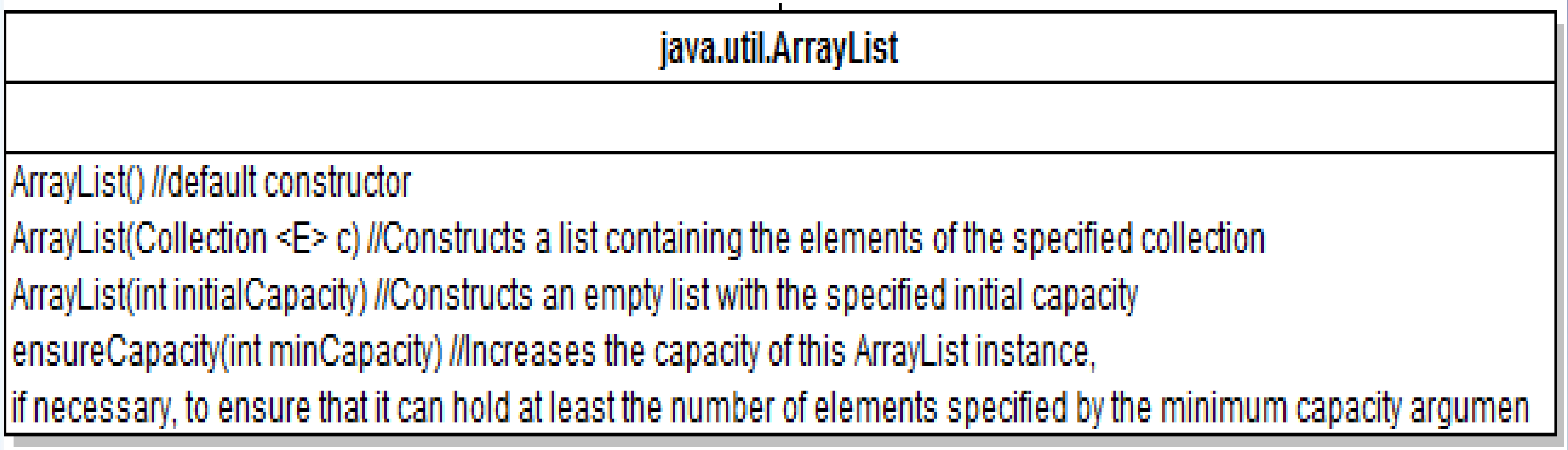
add(int index, E element):boolean //Inserts the specified element at the specified position
addAll(int index, Collection<E> c):boolean //Inserts all of the elements in the specified collection into this list at the specified position
get(int index):E //Returns the element at the specified position in this list.
indexOf(Object o): int //Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element
lastIndexOf(Object o):int //Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
remove(int index):E //Removes the element at the specified position in this list
set(int index, E element):E //Replaces the element at the specified position in this list with the specified element
subList(int fromIndex, int toIndex):List //Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.

ArrayList Class

List implementation based on an array of objects



Method “contains” based on comparing objects per method “equals”, complexity – ***O[N]***
Complexity of method “get” – ***O[1]***



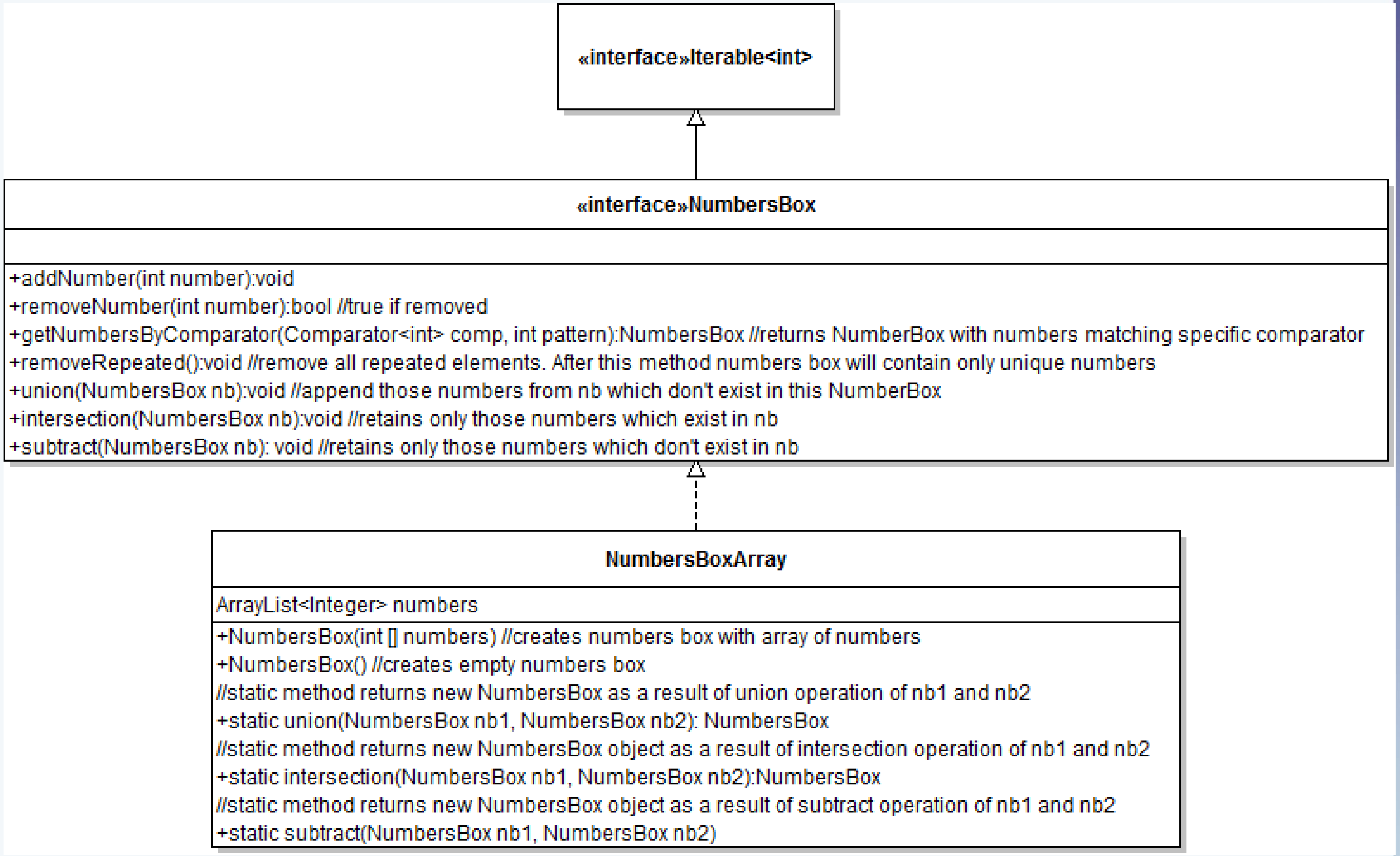
java.util.ArrayList

```
ArrayList() //default constructor
ArrayList(Collection <E> c) //Constructs a list containing the elements of the specified collection
ArrayList(int initialCapacity) //Constructs an empty list with the specified initial capacity
ensureCapacity(int minCapacity) //Increases the capacity of this ArrayList instance,
if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argumen
```

Exercise - Numbers Box Application

- Develop application including design and implementation allowing the following operations with numbers
  - Create random numbers in the given range
  - Display out all created numbers
  - Display out all numbers that may be divided by specific number with no remainder
  - Display out all numbers that great than specific number
  - Display out all numbers that great than specific number and may be divided by specific number with no remainder
  - Display out all numbers that don’t exist in the specific collection of numbers

Class NumbersBox



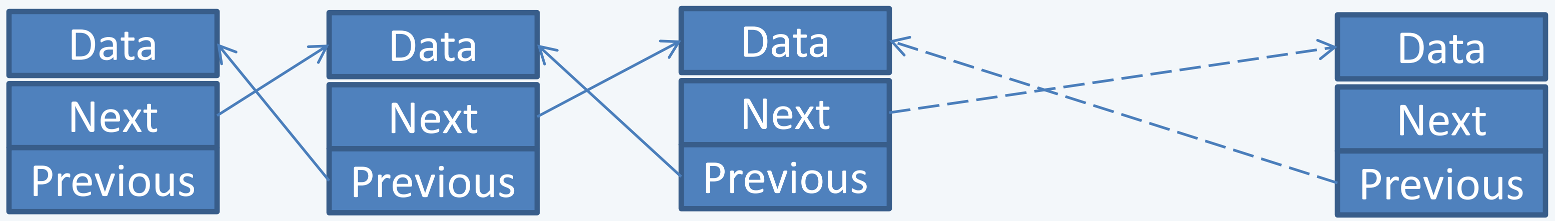
```
«interface»NumbersBox
+addNumber(int number):void
+removeNumber(int number):bool //true if removed
+getNumbersByComparator(Comparator<int> comp, int pattern):NumbersBox //returns NumberBox with numbers matching specific comparator
+removeRepeated():void //remove all repeated elements. After this method numbers box will contain only unique numbers
+union(NumbersBox nb):void //append those numbers from nb which don't exist in this NumberBox
+intersection(NumbersBox nb):void //retains only those numbers which exist in nb
+subtract(NumbersBox nb): void //retains only those numbers which don't exist in nb
```

```
NumbersBoxArray
ArrayList<Integer> numbers
+NumbersBox(int [] numbers) //creates numbers box with array of numbers
+NumbersBox() //creates empty numbers box
//static method returns new NumbersBox as a result of union operation of nb1 and nb2
+static union(NumbersBox nb1, NumbersBox nb2): NumbersBox
//static method returns new NumbersBox object as a result of intersection operation of nb1 and nb2
+static intersection(NumbersBox nb1, NumbersBox nb2):NumbersBox
//static method returns new NumbersBox object as a result of subtract operation of nb1 and nb2
+static subtract(NumbersBox nb1, NumbersBox nb2)
```

Linked List

- Two directional linked list
  - Element
    - Data
    - Reference to next
    - Reference to previous
  - Head – first element
  - Tail – last element
  - In each element, except tail, there is reference to a next element
  - In each element, except head, there is reference to a previous element

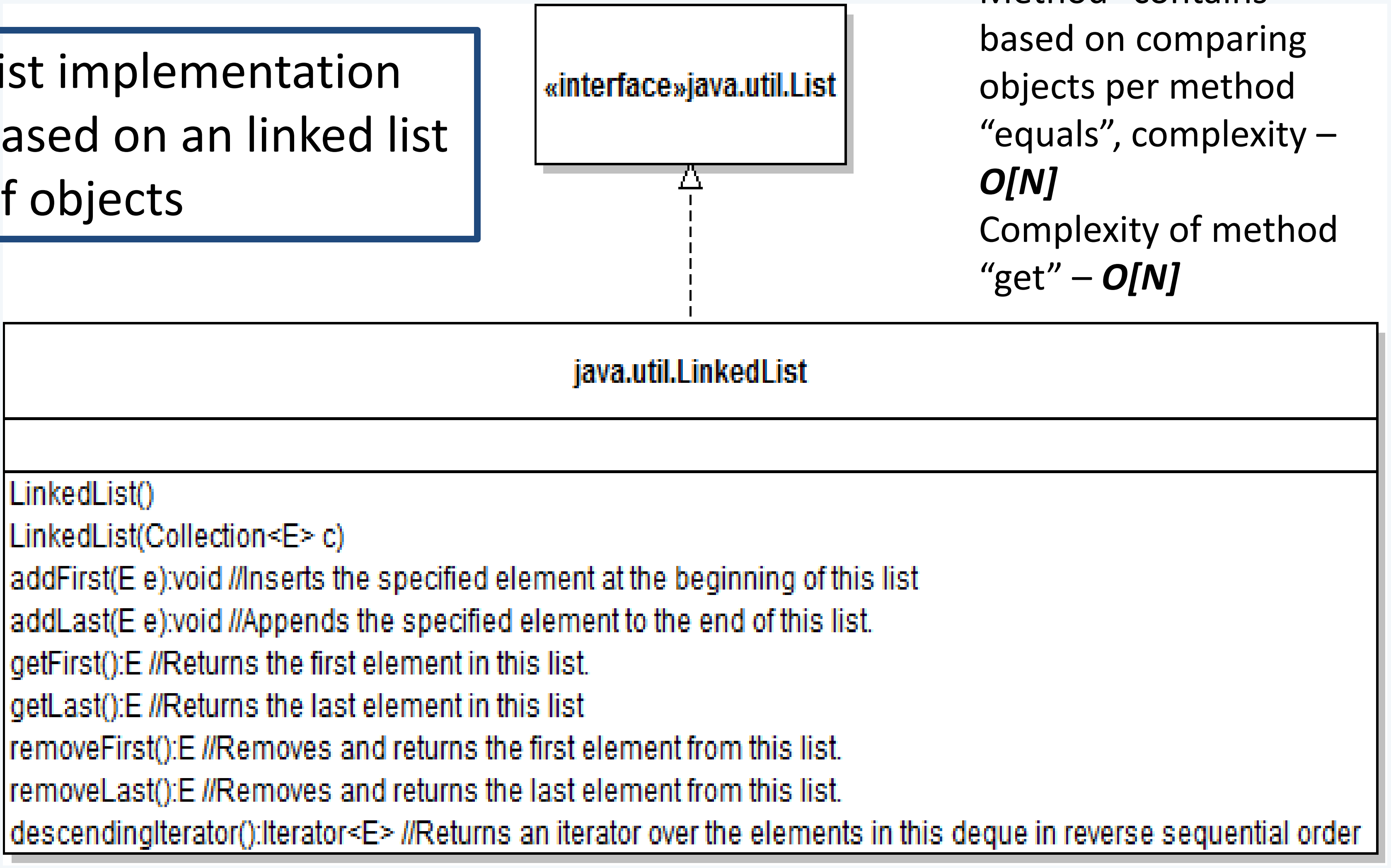
Quick adding with no additional memory
Quick removing with no additional memory
Slow access by index



LinkedList Class

List implementation based on an linked list of objects

Method “contains” based on comparing objects per method “equals”, complexity – ***O[N]***
Complexity of method “get” – ***O[N]***



```
java.util.LinkedList

LinkedList()
LinkedList(Collection<E> c)
addFirst(E e):void //Inserts the specified element at the beginning of this list
addLast(E e):void //Appends the specified element to the end of this list.
getFirst():E //Returns the first element in this list.
getLast():E //Returns the last element in this list
removeFirst():E //Removes and returns the first element from this list.
removeLast():E //Removes and returns the last element from this list.
descendingIterator():Iterator<E> //Returns an iterator over the elements in this deque in reverse sequential order
```

What is wrong ?

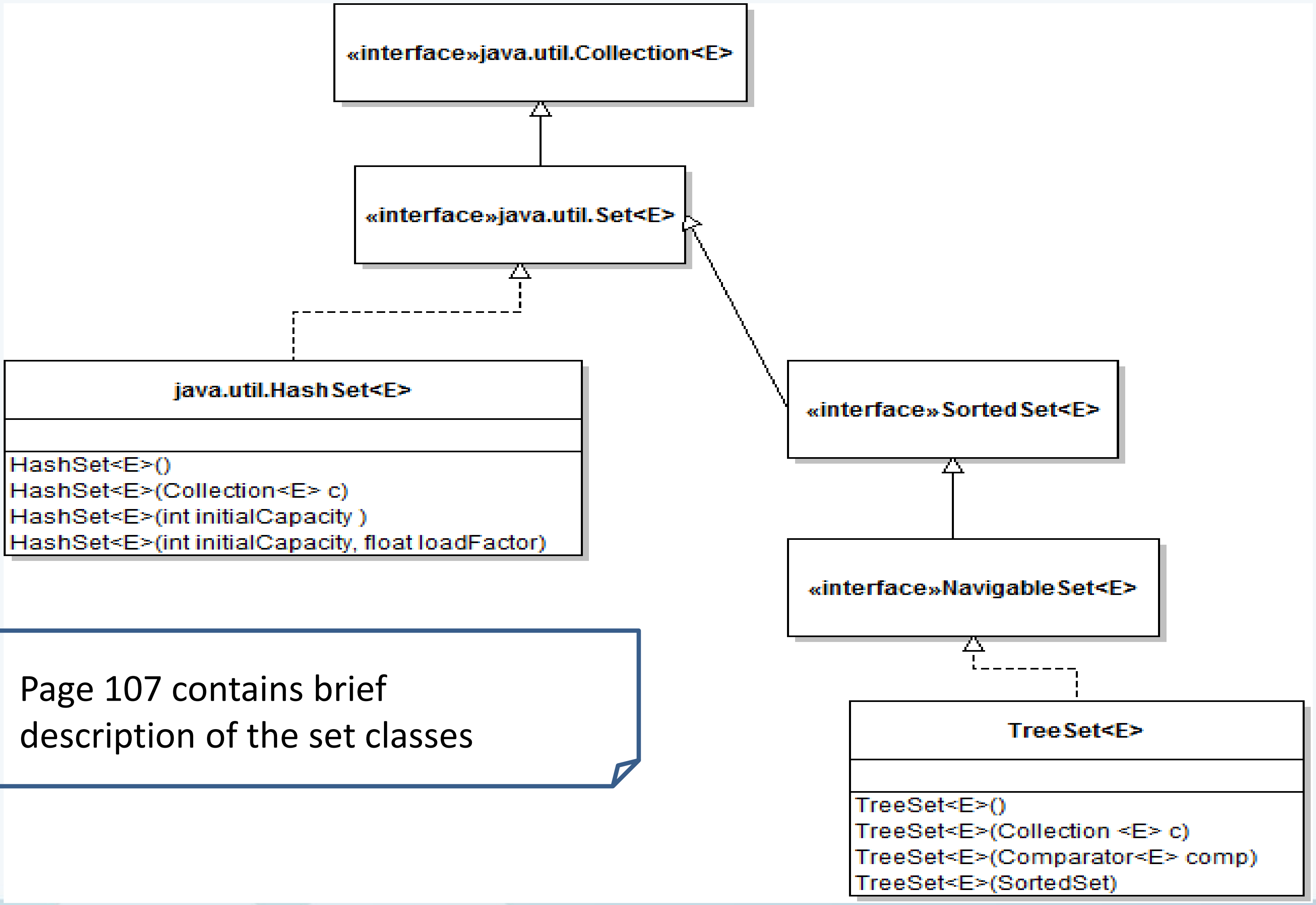
What's wrong in the below code:

```
LinkedList<Integer> linked = new
LinkedList<Integer>();
        for(int i=0; i<nElements; i++)
                System.out.println(linked.get(i));
```

Exercise-Array & Linked List

- NumbersBoxArray – class implementing interface NumbersBox based on java.util.ArrayList class
- NumbersBoxLinked – class implementing interface NumbersBox based on java.util.LinkedList class
- Two different solutions (different Eclipse projects). Any additional functions may be added including interface update
  - The same implementation of all interface methods in both classes
  - In the NumbersBoxArray the functions implemenmtation is based on get(int index) function call and in the NumbersBoxLinked the functions implementation is based on iterating
  - In both solutions the implementation of all the static methods should be the same

Set Collections


«interface»java.util.Collection<E>
«interface»java.util.Set<E>
java.util.HashSet<E>
HashSet<E>()
HashSet<E>(Collection<E> c)
HashSet<E>(int initialCapacity)
HashSet<E>(int initialCapacity, float loadFactor)
«interface»SortedSet<E>
«interface»NavigableSet<E>
TreeSet<E>
TreeSet<E>()
TreeSet<E>(Collection <E> c)
TreeSet<E>(Comparator<E> comp)
TreeSet<E>(SortedSet)

Page 107 contains brief description of the set classes

Set Implementation Classes

- ***TreeSet<E>***
  - Non-duplicated per “compare” function of E-type elements (There are no two or more objects “compare” function for which returns 0)
  - Complexities
    - Methods “contains”, “add”, ”remove” –***O[logN]*** per “compare” function
  - Sorted (guarantied iterating in ascending order per “compare” function
- ***HashSet<E>***
  - Non-duplicated per “hashCode” function of E-type elements (There are no two or more objects with the same hash code - “hashCode” method return)
  - Complexities
    - Methods “contains”, “add”, ”remove” –***O[1]*** per “hashCode” function

TEL-RAN

Sorted Set

Comparator<? super E>	comparator()Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.
E	first()Returns the first (lowest) element currently in this set.
SortedSet<E>	headSet(E toElement)Returns a view of the portion of this set whose elements are strictly less than toElement.
E	last()Returns the last (highest) element currently in this set.
SortedSet<E>	subSet(E fromElement, E toElement)Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
SortedSet<E>	tailSet(E fromElement)Returns a view of the portion of this set whose elements are greater than or equal to fromElement.

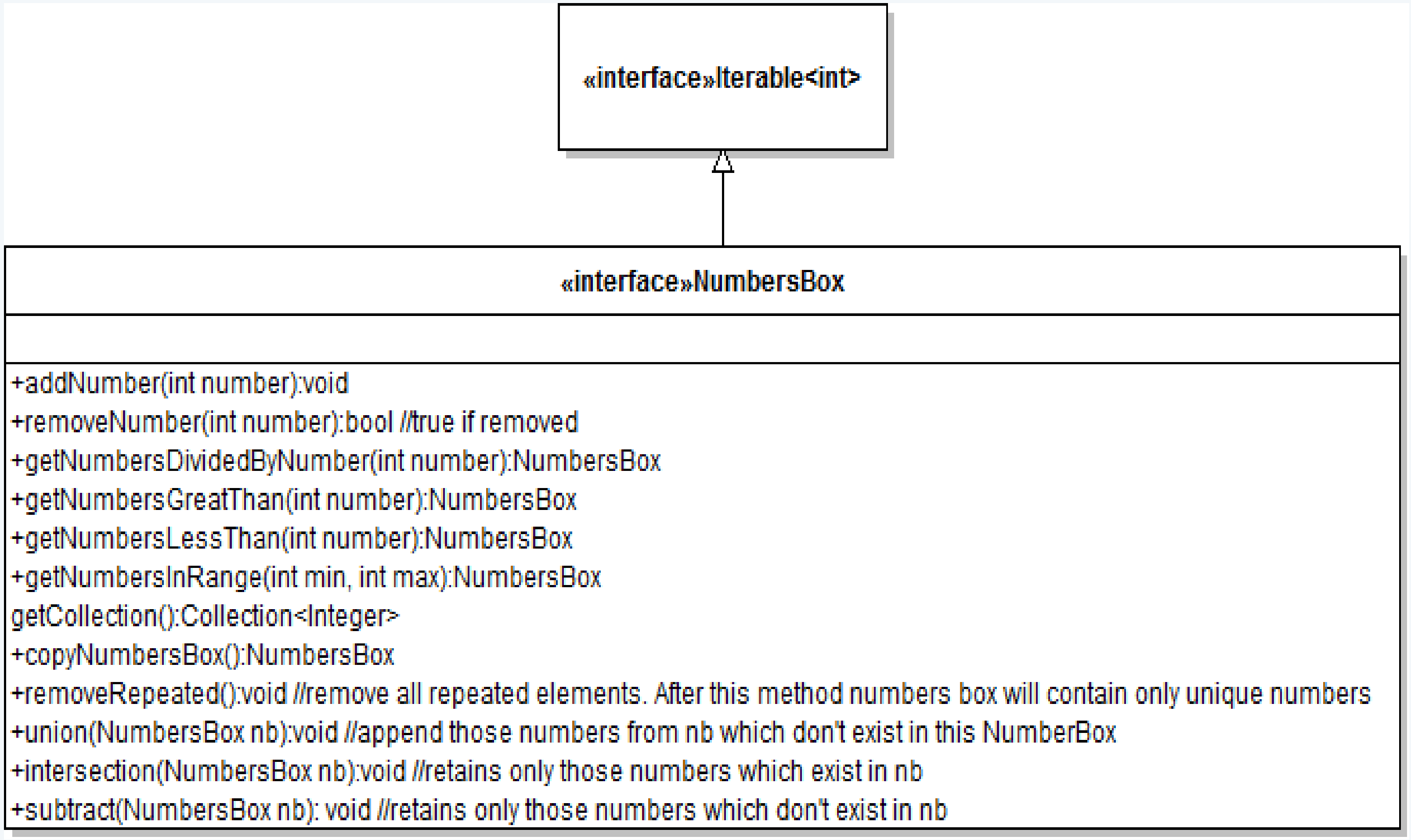
Navigable Set

E	ceiling(E e)Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
Iterator<E>	descendingIterator()Returns an iterator over the elements in this set, in descending order.
NavigableSet<E>	descendingSet()Returns a reverse order view of the elements contained in this set.
E	floor(E e)Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
NavigableSet<E>	headSet(E toElement, boolean inclusive)Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
E	higher(E e)Returns the least element in this set strictly greater than the given element, or null if there is no such element.
E	lower(E e)Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst()Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast()Retrieves and removes the last (highest) element, or returns null if this set is empty.
NavigableSet<E>	subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)Returns a view of the portion of this set whose elements range from fromElement to toElement.
NavigableSet<E>	tailSet(E fromElement, boolean inclusive)Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

Exercise – NumbersBox based on Collection

- Develop classes NumbersBoxTree, NumbersBoxArray, NumbersBoxLinked implementing interface on the next page
- Develop controller with functionality described on the page 100 and additionally the following methods
  - Display out all elements that less than the specific number
  - Display out all elements great or equal of the first specific number and less than the second specific number (Existence in a range)
- Define time of the controller running for each implementation classes

Interface NumbersBox for Exercise



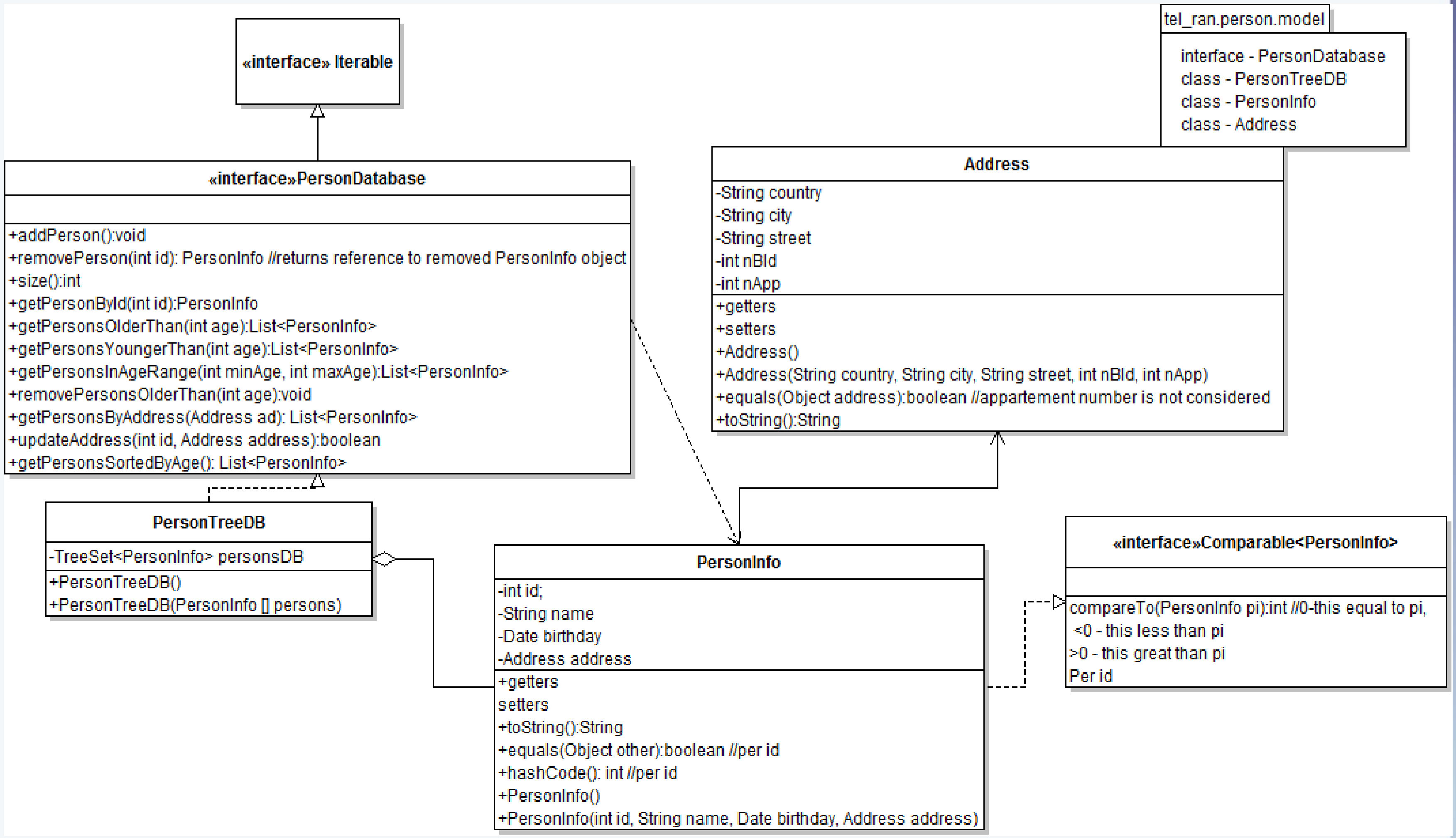
«interface»NumbersBox

+addNumber(int number):void
+removeNumber(int number):bool //true if removed
+getNumbersDividedByNumber(int number):NumbersBox
+getNumbersGreatThan(int number):NumbersBox
+getNumbersLessThan(int number):NumbersBox
+getNumbersInRange(int min, int max):NumbersBox
getCollection():Collection<Integer>
+copyNumbersBox():NumbersBox
+removeRepeated():void //remove all repeated elements. After this method numbers box will contain only unique numbers
+union(NumbersBox nb):void //append those numbers from nb which don't exist in this NumberBox
+intersection(NumbersBox nb):void //retains only those numbers which exist in nb
+subtract(NumbersBox nb): void //retains only those numbers which don't exist in nb

Person Application on TreeSet

- Page 113 – Class diagram for model package
- Page 114 – Class diagram for view package
- Page 115 – Class diagram for controller package
- Page 116 – Activity diagram for function test

Person Application Model



```
tel_ran.person.model
  interface - PersonDatabase
  class - PersonTreeDB
  class - PersonInfo
  class - Address
```

```
«interface» Iterable
```

```
«interface»PersonDatabase
----
+addPerson():void
+removePerson(int id): PersonInfo //returns reference to removed PersonInfo object
+size():int
+getPersonById(int id):PersonInfo
+getPersonsOlderThan(int age):List<PersonInfo>
+getPersonsYoungerThan(int age):List<PersonInfo>
+getPersonsInAgeRange(int minAge, int maxAge):List<PersonInfo>
+removePersonsOlderThan(int age):void
+getPersonsByAddress(Address ad): List<PersonInfo>
+updateAddress(int id, Address address):boolean
+getPersonsSortedByAge(): List<PersonInfo>
```

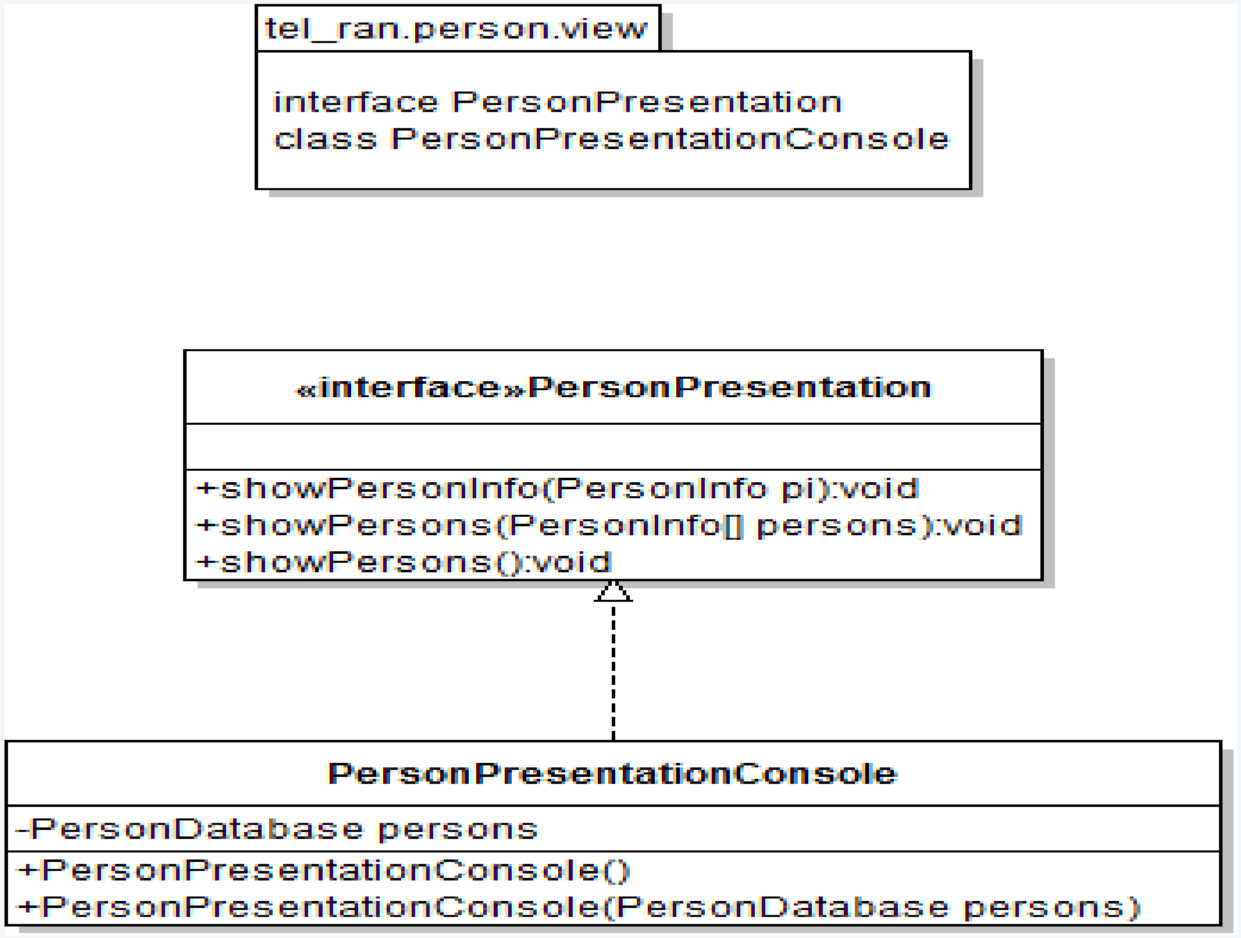
```
Address
-String country
-String city
-String street
-int nBld
-int nApp
+getters
+setters
+Address()
+Address(String country, String city, String street, int nBld, int nApp)
+equals(Object address):boolean //appartement number is not considered
+toString():String
```

```
PersonTreeDB
-TreeSet<PersonInfo> personsDB
+PersonTreeDB()
+PersonTreeDB(PersonInfo [] persons)
```

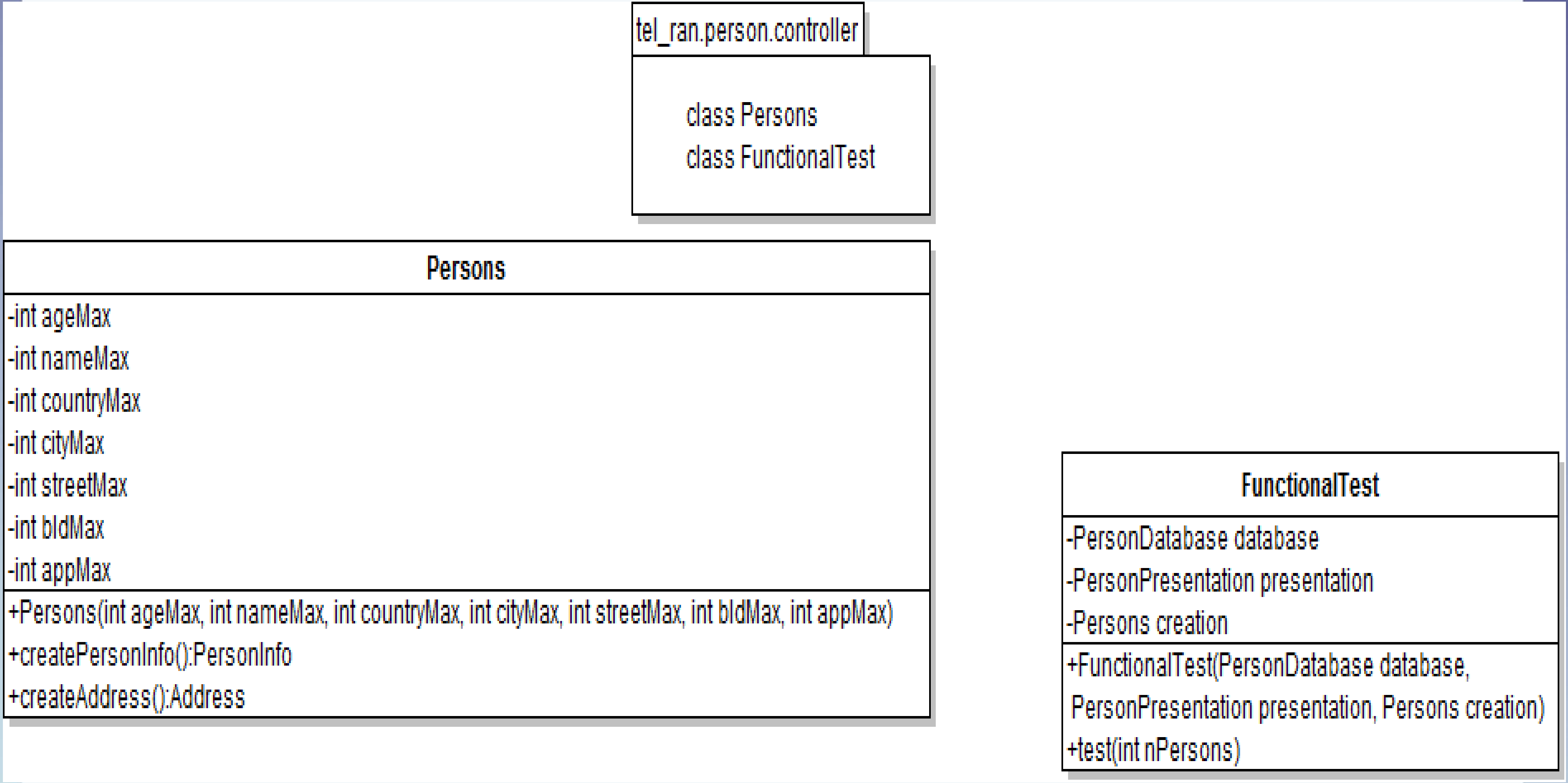
```
PersonInfo
-int id;
-String name
-Date birthday
-Address address
+getters
setters
+toString():String
+equals(Object other):boolean //per id
+hashCode(): int //per id
+PersonInfo()
+PersonInfo(int id, String name, Date birthday, Address address)
```

```
«interface»Comparable<PersonInfo>
----
compareTo(PersonInfo pi):int //0-this equal to pi,
 <0 - this less than pi
>0 - this great than pi
Per id
```

Person Application View


tel_ran.person.view
interface PersonPresentation
class PersonPresentationConsole
«interface»PersonPresentation
+showPersonInfo(PersonInfo pi):void
+showPersons(PersonInfo[] persons):void
+showPersons():void
PersonPresentationConsole
-PersonDatabase persons
+PersonPresentationConsole()
+PersonPresentationConsole(PersonDatabase persons)

Person Application Controller



Persons

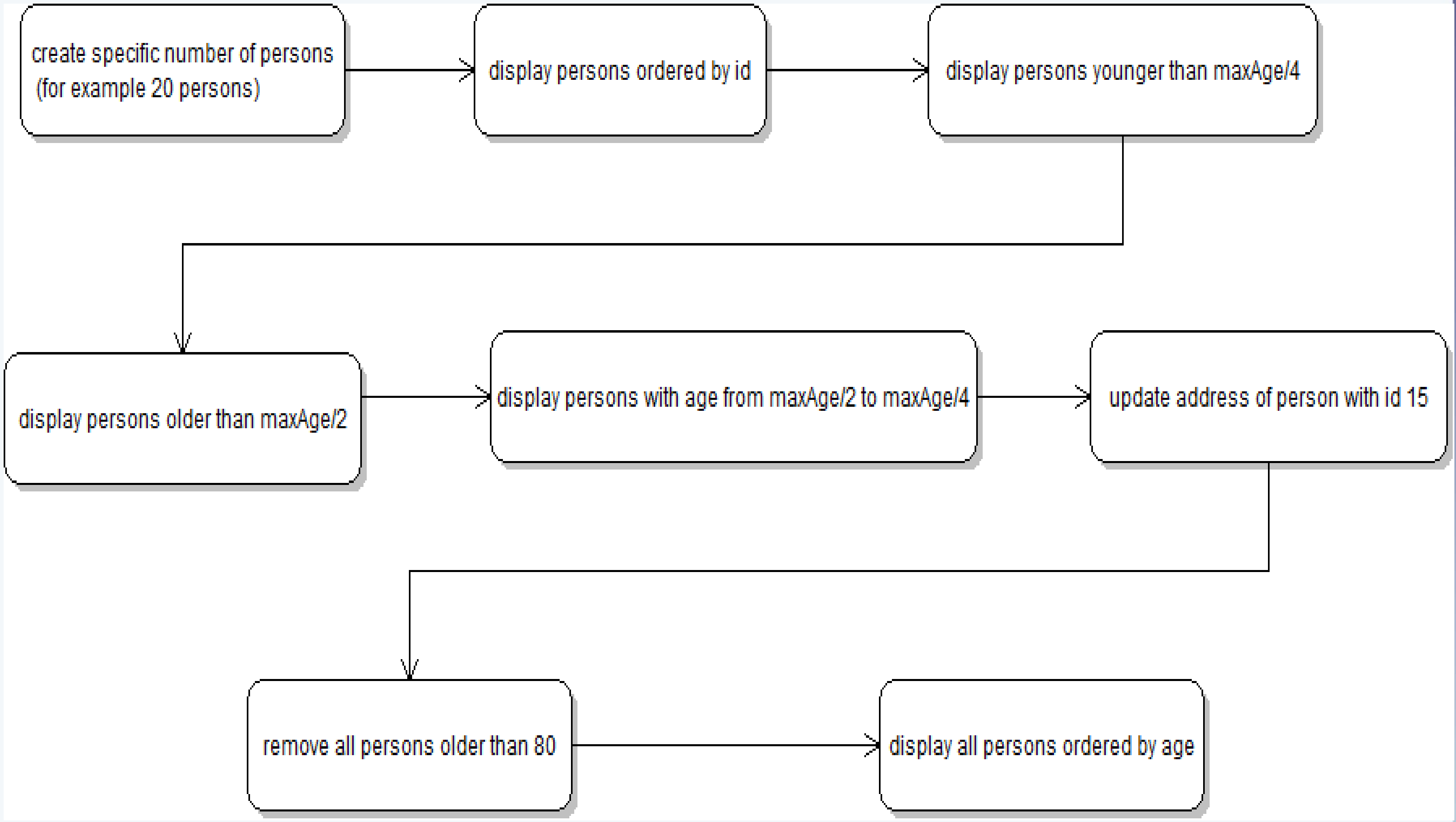
```
-int ageMax
-int nameMax
-int countryMax
-int cityMax
-int streetMax
-int bldMax
-int appMax
+Persons(int ageMax, int nameMax, int countryMax, int cityMax, int streetMax, int bldMax, int appMax)
+createPersonInfo():PersonInfo
+createAddress():Address
```

FunctionalTest

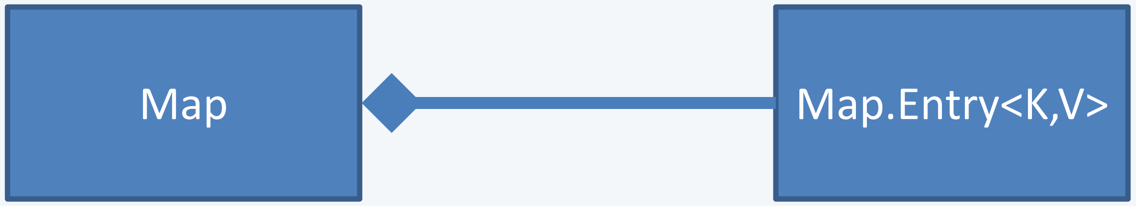
```
-PersonDatabase database
-PersonPresentation presentation
-Persons creation
+FunctionalTest(PersonDatabase database,
 PersonPresentation presentation, Persons creation)
+test(int nPersons)
```

TEL-RAN

Test Function Activity Diagram


create specific number of persons (for example 20 persons)
display persons ordered by id
display persons younger than maxAge/4
display persons older than maxAge/2
display persons with age from maxAge/2 to maxAge/4
update address of person with id 15
remove all persons older than 80
display all persons ordered by age

Map



- Mapping keys to values.
  - Map<K, V> K-type of key, V-type of value
- No duplicate keys
- It models the mathematical *function* abstraction. *V=F(K)*
- *Basic operation*
  - put, get, remove, containsKey, containsValue, size, empty
- Bulk operations
  - putAll , clear
- Collection views
  - keySet, entrySet, values

Map Interface

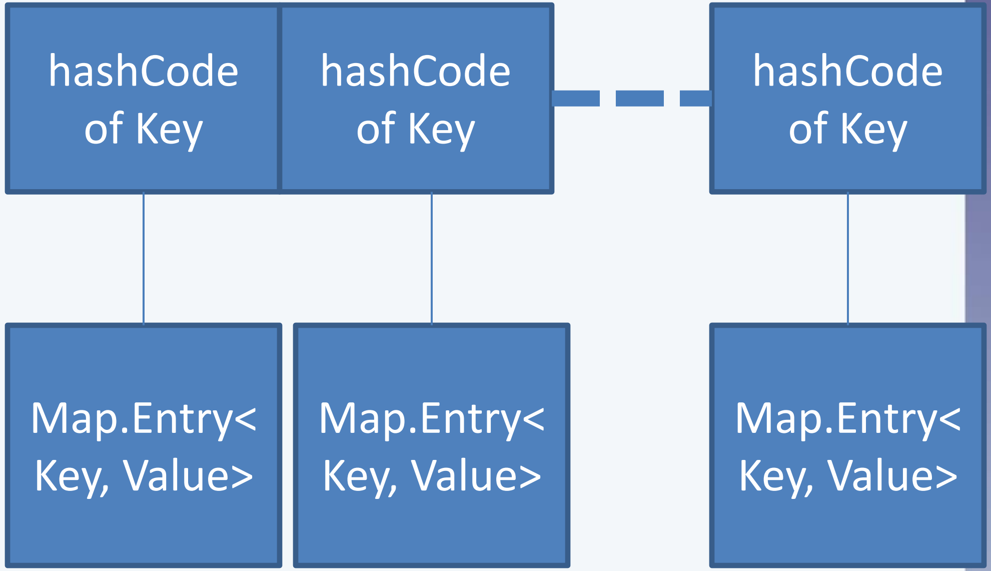
void	clear()Removes all of the mappings from this map
boolean	containsKey(Object key)Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value)Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	entrySet()Returns a Set view of the mappings contained in this map.
V	get(Object key)Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean	isEmpty()Returns true if this map contains no key-value mappings.
Set<K>	keySet()Returns a Set view of the keys contained in this map.
V	put(K key, V value)Associates the specified value with the specified key in this map (optional operation).
void	putAll(Map<K,V> m)Copies all of the mappings from the specified map to this map
V	remove(Object key)Removes the mapping for a key from this map if it is present).
int	size()Returns the number of key-value mappings in this map.
Collection<V>	values()Returns a Collection view of the values contained in this map.

Map.Entry<K,V>

boolean	equals(Object o)Compares the specified object with this entry for equality.
K	getKey()Returns the key corresponding to this entry.
V	getValue()Returns the value corresponding to this entry.
int	hashCode()Returns the hash code value for this map entry.
V	setValue(V value)Replaces the value corresponding to this entry with the specified value (optional operation).

HashMap

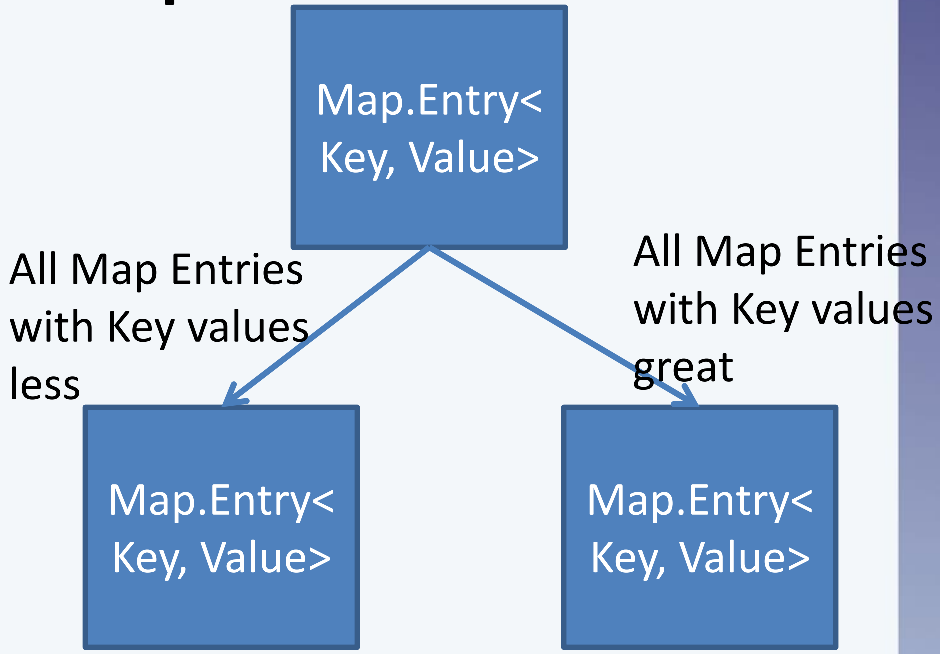
- Unsorted collection of the Map Entries
- Complexity of value access via key is ***O[1]***



HashMap()Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
HashMap(int initialCapacity)Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
HashMap(int initialCapacity, float loadFactor)Constructs an empty HashMap with the specified initial capacity and load factor.
HashMap(**Map**<**K**, **V**> m)Constructs a new HashMap with the same mappings as the specified Map.

TreeMap

- Sorted collection of the Map Entries according to the Key values
- Complexity of value access is ***O[logN]***

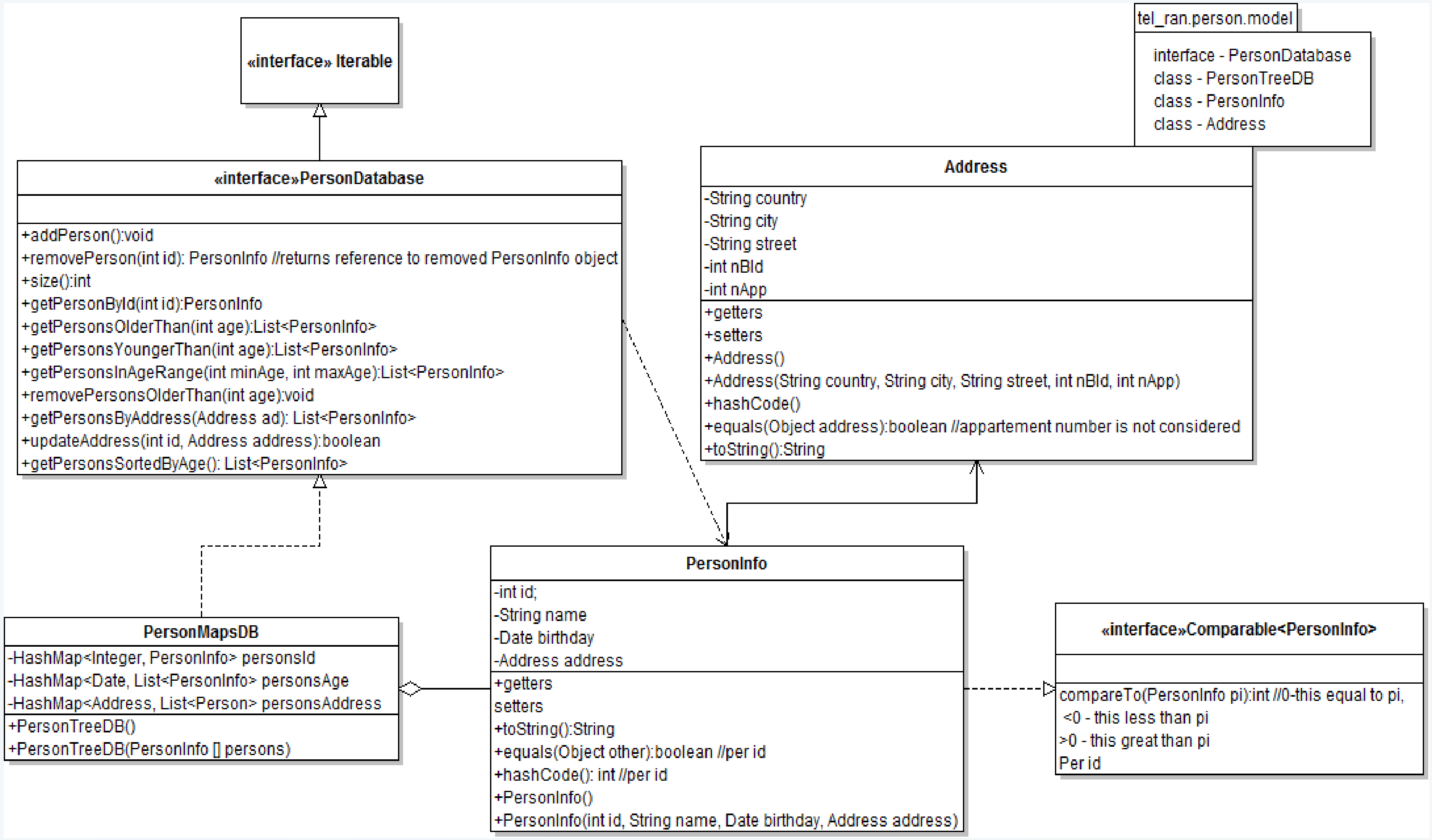

Map.Entry< Key, Value>
All Map Entries with Key values less
All Map Entries with Key values great
Map.Entry< Key, Value>
Map.Entry< Key, Value>

TreeMap()Constructs a new, empty tree map, using the natural ordering of its keys.
TreeMap(Comparator< K> comparator)Constructs a new, empty tree map, ordered according to the given comparator.
TreeMap(Map<K, V> m)Constructs a new tree map containing the same mappings as the given map, ordered according to the *natural ordering* of its keys.
TreeMap(SortedMap<K, V> m)Constructs a new tree map containing the same mappings and using the same ordering as the specified sorted map.

TreeMap as Navigable-Sorted Implementation (Most Used Methods)

SortedMap<K,V>	headMap(K toKey)Returns a view of the portion of this map whose keys are strictly less than toKey.
NavigableMap<K,V>	headMap(K toKey, boolean inclusive)Returns a view of the portion of this map whose keys are less than (or equal to, if inclusive is true) toKey.
NavigableMap<K,V>	subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)Returns a view of the portion of this map whose keys range from fromKey to toKey.
SortedMap<K,V>	subMap(K fromKey, K toKey)Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive.
SortedMap<K,V>	tailMap(K fromKey)Returns a view of the portion of this map whose keys are greater than or equal to fromKey.
NavigableMap<K,V>	tailMap(K fromKey, boolean inclusive)Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.

Exercise – Optimal Person Model Implementation



«interface» Iterable
tel_ran.person.model
interface - PersonDatabase
class - PersonTreeDB
class - PersonInfo
class - Address
«interface»PersonDatabase
+addPerson():void
+removePerson(int id): PersonInfo //returns reference to removed PersonInfo object
+size():int
+getPersonById(int id):PersonInfo
+getPersonsOlderThan(int age):List<PersonInfo>
+getPersonsYoungerThan(int age):List<PersonInfo>
+getPersonsInAgeRange(int minAge, int maxAge):List<PersonInfo>
+removePersonsOlderThan(int age):void
+getPersonsByAddress(Address ad): List<PersonInfo>
+updateAddress(int id, Address address):boolean
+getPersonsSortedByAge(): List<PersonInfo>
Address
-String country
-String city
-String street
-int nBld
-int nApp
+getters
+setters
+Address()
+Address(String country, String city, String street, int nBld, int nApp)
+hashCode()
+equals(Object address):boolean //appartement number is not considered
+toString():String
PersonMapsDB
-HashMap<Integer, PersonInfo> personsId
-HashMap<Date, List<PersonInfo> personsAge
-HashMap<Address, List<Person> personsAddress
+PersonTreeDB()
+PersonTreeDB(PersonInfo [] persons)
PersonInfo
-int id;
-String name
-Date birthday
-Address address
+getters
setters
+toString():String
+equals(Object other):boolean //per id
+hashCode(): int //per id
+PersonInfo()
+PersonInfo(int id, String name, Date birthday, Address address)
«interface»Comparable<PersonInfo>
compareTo(PersonInfo pi):int //0-this equal to pi,
<0 - this less than pi
>0 - this great than pi
Per id

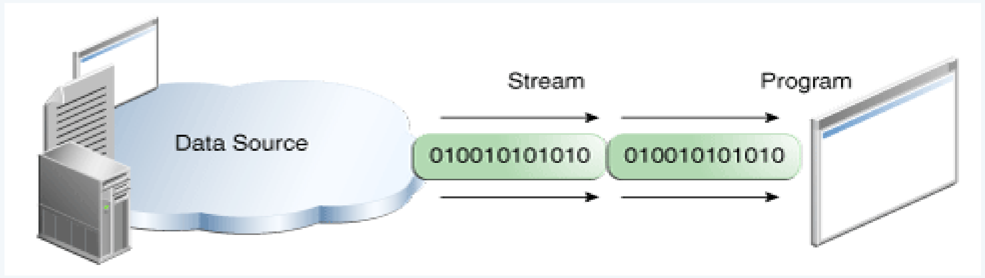
Exercise – Natural Numbers

- Develop class that allows getting in the most effective way the following information about the natural numbers existing in the range [1, 50000]
  - Factors for specific number (a factor is a number by which the specific number may be divided with no remainder)
  - The greatest common divisor for a list of the specific numbers
  - Whether the specific number is the prime one

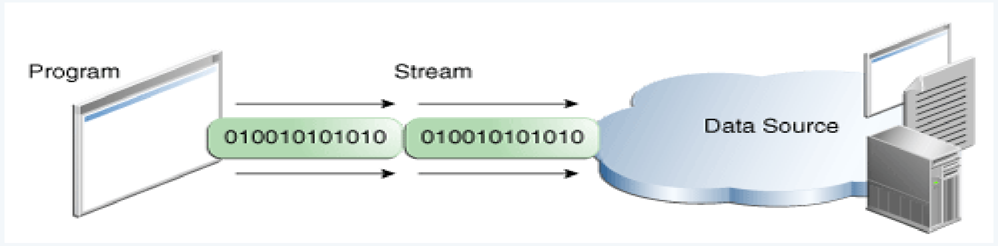
Exercise – Life Model Application

- Beginning from current date, ending up 1/1/2500
- Each day 5-10 people are born
- Each day 2-3 people are died (everyone who was born should die). Below is the person mortality distribution depending on the age
  - 80-85 with probability 30%
  - 86-88 with probability 60%
  - 89-92 with probability 90%
  - 93-100 with probability 96%
  - 101-110 with probability 99%
- Display out the following
  - How many people will be alive by 1/1/2500
  - Person information of all people who got to age older than 100 years during whole running time period up to 1/1/2500
  - Person information of all people who will be alive by 1/1/2500 older than 90 years

Input-Output Streams



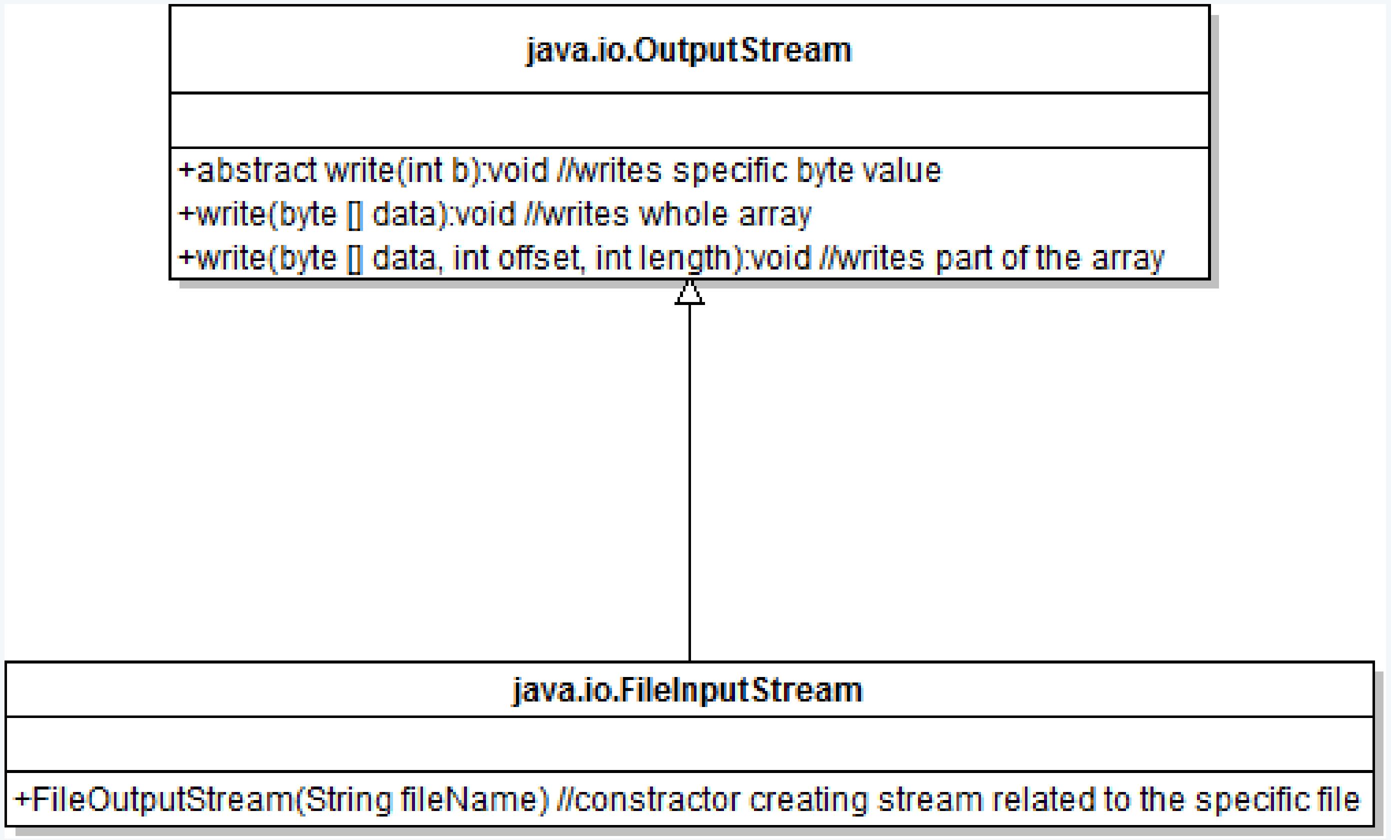
Input Stream - read data from a source, one item at a time



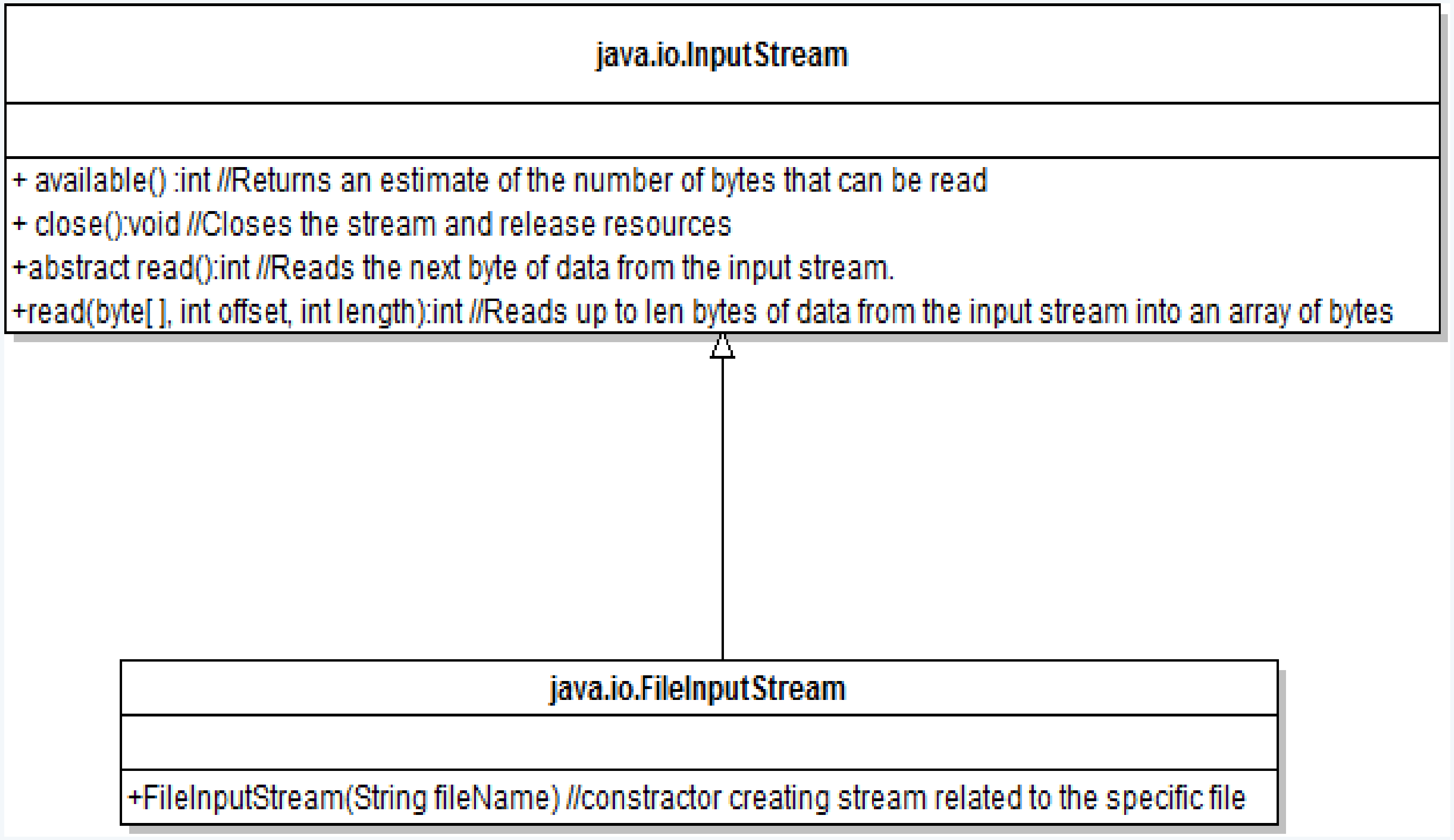
Output Stream - write data to a destination, one item at time

- Stream is a sequence of data
- An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- Streams support many different kinds of data, including simple bytes, primitive data types, and objects

Byte Streams – Output Stream


java.io.OutputStream
+abstract write(int b):void //writes specific byte value
+write(byte [] data):void //writes whole array
+write(byte [] data, int offset, int length):void //writes part of the array
java.io.FileInputStream
+FileOutputStream(String fileName) //constractor creating stream related to the specific file

Byte Streams-Input Stream


java.io.InputStream
+ available() :int //Returns an estimate of the number of bytes that can be read
+ close():void //Closes the stream and release resources
+abstract read():int //Reads the next byte of data from the input stream.
+read(byte[ ], int offset, int length):int //Reads up to len bytes of data from the input stream into an array of bytes
java.io.FileInputStream
+FileInputStream(String fileName) //constractor creating stream related to the specific file

Exercise – Byte Streams

- Write random amount of the integers from 50 to 100 to a file as strings (text). Each number is in the range from 10 to 100
- Read numbers from the file created in the above item and compute the sum of all numbers

Note: To get array of string’s tokens separated each from other by the blank

```
        String ar[]=str.split(“ “);
```

Character Streams

- Uses characters according to the default encoding like class String
- Classes
  - ***Writer*** abstract class for writing characters into stream
    - ***FileWriter*** – for writing characters into file
    - ***FileWriter*** constructors

    FileWriter(String fileName)

    FileWriter(String fileName, boolean appendFlag)
  - ***Reader*** abstract class for reading characters from stream
    - ***FileReader*** – for reading characters from a file

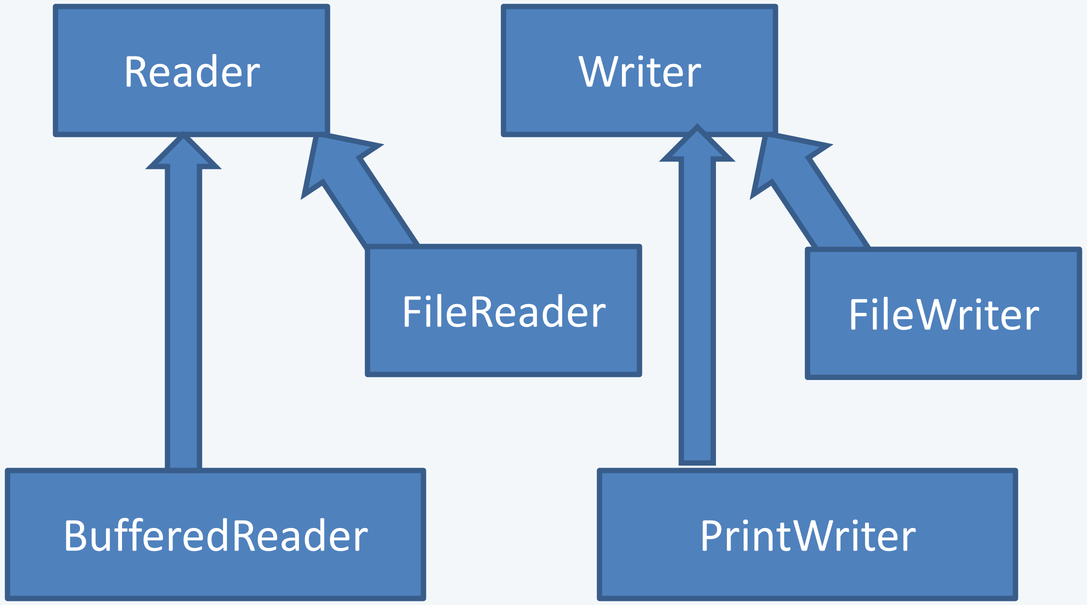
    FileReader (String fileName) - constructor

Line Streams

- Class PrintWriter character oriented
  - Constructors
    - ***PrintWriter(Writer w); PrintWriter(String fileName);***
  - Methods ***print*** and ***println***
- Class ***PrintStream*** byte oriented
  - Constructors
    - ***PrintStream(OutputStream out); PrintStream(String fileName);***
  - Methods ***print*** and ***println***
- Class ***BufferedReader*** character oriented
  - ***BufferedReader(Reader in); -*** constructor

  BufferedReader in=new BufferedReader(new FileReader (“file.txt”));
  - Method ***readLine()*** reading string from a stream

Classes Hierarchy Char Oriented for Files



Reader – Char oriented abstract Input Stream (reads characters)
Writer – Char oriented abstract Output Stream (writes characters)
FileReader – Char oriented class for reading characters from file
FileWriter – Char oriented class for writing characters to file
BufferedReader – Char oriented class with method readLine() for reading string from stream. Constructor of this class receives reference to concrete stream, for example, file or console
PrintWriter – Char oriented class with method println(String str) for writing string to stream

Console Input

- Console is a virtual device for input/output operations
  - Default input from console related to the keyboar
- ***System.in*** is object of ***InputStream*** class related to the console
  - To use the method readLine() from InputStream we should apply class InputStreamReader allowing conversion from byte oriented stream to character oriented

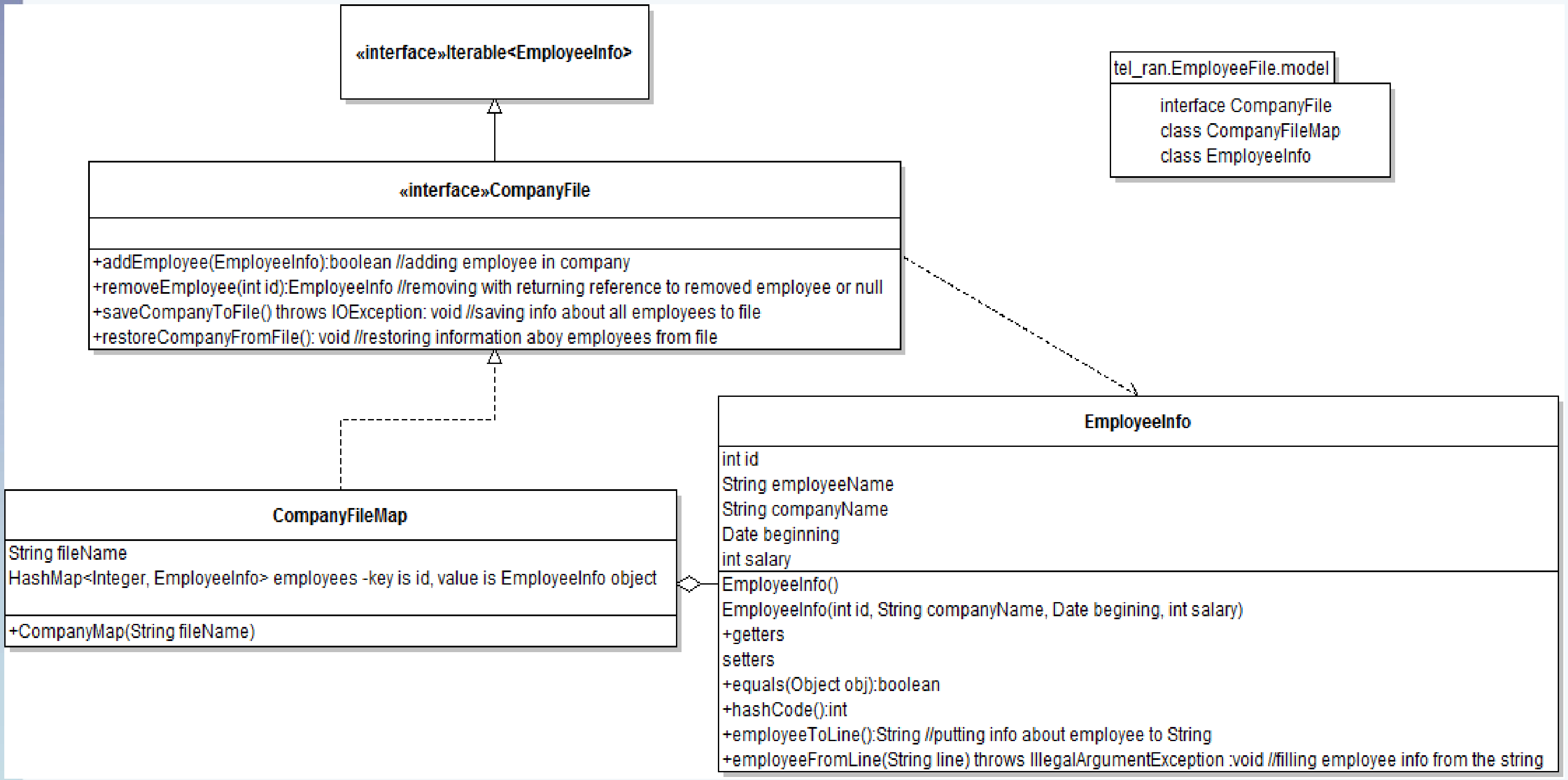
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

  - Waiting for entering “new line” from console (key “enter”)
    - Each readLine() call causes waiting for entering line

Exercise – Company File Application Line Oriented Streams

- Model – slide# 134
- View – slide# 135
- Controller
  - Two application classes:
    1. Creating company
       - Creating company based on info from console input
       - Saving company to file “company.txt” inside the project’s directory
    2. Restoring company
       - Restoring company from the file “company.txt”
       - Removing one any employee
       - Displaying all employees

Employee File Model



«interface»CompanyFile

+addEmployee(EmployeeInfo):boolean //adding employee in company
+removeEmployee(int id):EmployeeInfo //removing with returning reference to removed employee or null
+saveCompanyToFile() throws IOException: void //saving info about all employees to file
+restoreCompanyFromFile(): void //restoring information aboy employees from file

CompanyFileMap

String fileName
HashMap<Integer, EmployeeInfo> employees -key is id, value is EmployeeInfo object

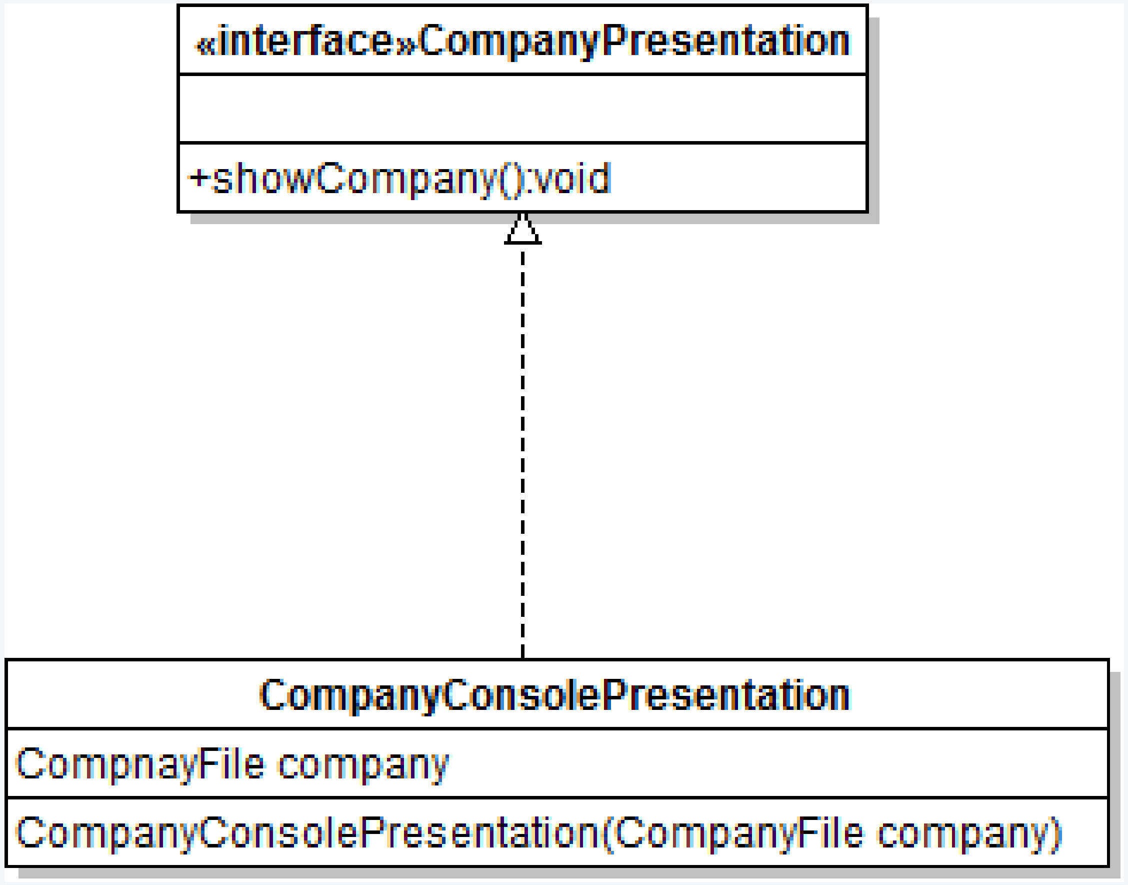
+CompanyMap(String fileName)

EmployeeInfo

int id
String employeeName
String companyName
Date beginning
int salary

EmployeeInfo()
EmployeeInfo(int id, String companyName, Date begining, int salary)
+getters
setters
+equals(Object obj):boolean
+hashCode():int
+employeeToLine():String //putting info about employee to String
+employeeFromLine(String line) throws IllegalArgumentException :void //filling employee info from the string

Employee File View


«interface»CompanyPresentation
+showCompany():void
CompanyConsolePresentation
CompnayFile company
CompanyConsolePresentation(CompanyFile company)

Exercise-Calculator from File

- Write application that computes arithmetic operations (+,-,/,*) with data from an input text file and creates an output text file with results
  - Format of the input file's line

    <float number> <float number> <operation symbol>
    - Number of lines is not defined
  - Output file
    - First line

      Operand1 Operand2 Operation Result
    - Other lines

      <float number> <float number> <operation symbol> <float number>|****

      Stars mark wrong input data

Example for Calculator from File Application

- Input File

```
25.5 12 +
5 10 *
Abc 12 /
2 4 &
```

- Output File

```
Operand1 Operand2 Operation Result
25.5 12 + 37.5
5 10 * 50
Abc 12 / ****
2 4 & ****
```

Exercise – English Dictionary

- Write application that creates English words dictionary from full text of the “War and Peace” book, and performs pointed below actions
  - Full text exits in ***Dropbox\Java-6\Data\WarPeace.txt***
  - The dictionary should allow the following functionality:
    - Console input: one letter
      - Console output: all words starting with the entered letter (10 words on the output line)
    - Console input: word
      - Console output: If the word exists there will not be any output, otherwise output should contain all existing words starting with the same letters (60% from the word’s length)as in the entered. For example: input: ***bex***, output: ***best bed better benefit …***

Text Line and Words

- What is “word”
  - a-z | A-Z … One or many letters
- Splitting text on words

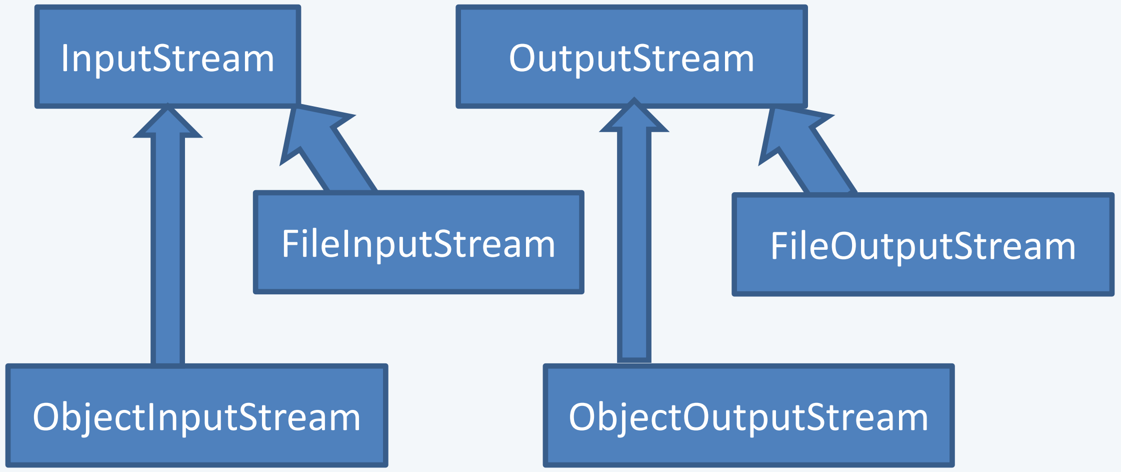
String[] words=line.split(“[^a-zA-Z]”);

  - Method split of the class String gets regular expression of the delimiters (symbols separating string tokens)

Object Oriented Streams

- Kind of byte oriented streams
- Input Stream
  - Class ***ObjectInputStream*** with method ***readObject()*** returning object of any class
- Output Stream
  - Class ObjectOutputStream with method ***writeObject(Object obj)***
- Object should be instance of any class implementing interface ***Serializable***
  - Interface ***Serializable*** is Java standard interface which doesn't require any additional implementation
  - All class fields that are references to other objects should designate classes implementing interface Serializable

Classes Hierarchy Byte and Object Oriented for Files



- Constructors of Object oriented streams take reference to appropriate byte oriented streams.
  - Example of creating an object oriented output stream for writing object to file “objectsFile”

ObjectOutputStream out=new ObjectOutputStream(new FileOutputStream(“objectsFile”));
out.writeObject(object);

  - Example of creating an object oriented input stream for reading object from file “objectsFile”

ObjectInputStream in=new ObjectInputStream(new FileInputStream(“objectsFile”));
Object obj=in.readObject();

Exercise – Company File Application Object Oriented Streams

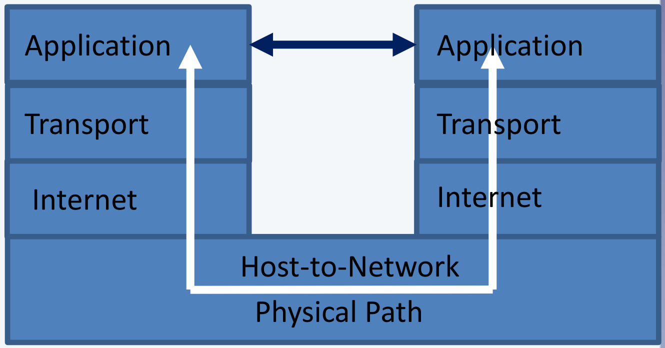
- Develop class CompanyMapObjectStream implementing the interface CompanyFile (slide #135)
  - The same implementation as CompanyFileMap but with saving to and restoring from Object Stream
- Update application described on slide # 134 using both Object Oriented and Line Oriented Streams and compare time values for saving and restoring

IP Networking

Network Layers

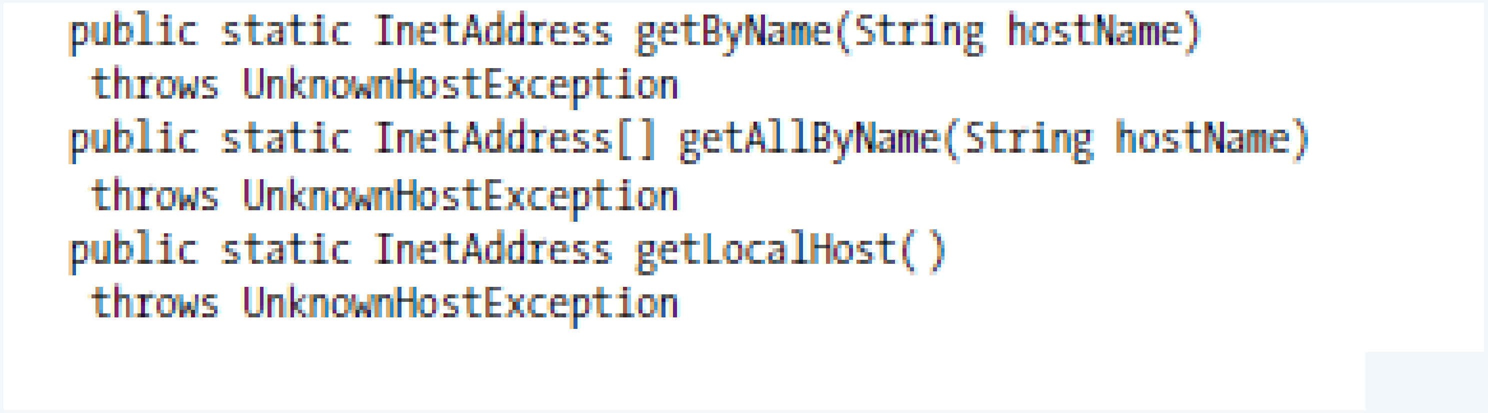
- Network
  - Collection of computers and other devices that can send data to or receive data from each other
- Node
  - Device on Network
- Host
  - Node which is a general-purpose computer

TCP/IP four-layers model



Application Layer – HTTP, FTP, etc.
Transport Layer – TCP, UDP
Internet – IPv4, IPv6
Host-t-Network – Ethernet, WiFi, etc.

IP Address and Port

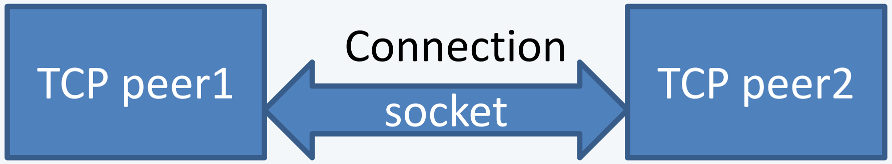


```
public static InetAddress getByName(String hostName)
 throws UnknownHostException
public static InetAddress[] getAllByName(String hostName)
 throws UnknownHostException
public static InetAddress getLocalHost( )
 throws UnknownHostException
```

- Each host has IP address and often host name
- IP Address is represented by InetAddress class
- InetAddress class has private constructor
- Object of the class InetAddress may be got by the above methods
  - ***hostname*** parameters in the method getByName may be string containing either host name or IP address
- Each communication application running on a host has a port number identifying socket for sending to and receiving data from another process or host
  - The port number shall be unique per host and protocol (UDP or TCP)

Transport Control Protocol (TCP)

- Connection oriented
  - 3-way handshake connection establishment
- Reliable
  - State machine providing control for data transferring (Order and Checksum)



TEL-RAN

TCP Client

- Connection establishment with server

  Socket socket = new Socket(<hostname>, <port number>);

  - Two possible exceptions :
    - ***UnknownHostException***
    - ***IOException***
- Reading from socket

  socket.getInputStream()

  - Filtering to InputStreamReader
  - For efficiency and working with strings filtering to ***BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));***
- Writing to socket

  socket.getOutputStream()

  Auto flushing

  - ASCII protocol
    - Filtering to ***PrintWriter out= new PrintWriter(socket.getOutputStream(), true);***
  - Binary protocol
    - Filtering to OutputStreamWriter and BufferedWriter ***out=new BufferedWriter (new OutputStreamWriter (socket.getOutputStream()));***
- Close streams and connection

  in.close(); out.close(); socket.close();

TCP Server

- Listening to connection from a client on the given port
  - Socket for listening with IOException

    ServerSocket ss =new ServerSocket(<port number>);
  - Connection accepting

    Socket socket = ss.accept();
- Data exchange with a client through the accepted and created dedicated socket as it has been described in TCP client
  socket.getInputStream()/socket.getOutputStream()

Exercise – TCP Client/Server

- Implement simple Echo protocol
  - Client gets string from the console and sends it to server
  - Server responses a message containing the same string concatenated with “response from server”
  - Client displays the received message on the console
- Implement the following protocol Server has array of the “clever” messages like “don’t forget drink water”
  - Client may send the following requests
    - “getNumberOfMessages”
    - “getMessage”
  - Server on the request “getNumberOfMessages” sends the client the length of the messages array
  - Server on the request “getMessage” sends the client random message
  - Server on an unknown requests closes connection with the client
  - Using that protocol write client printing out all non duplicated “clever” messages the server owns

Exercise – Calculator on Server

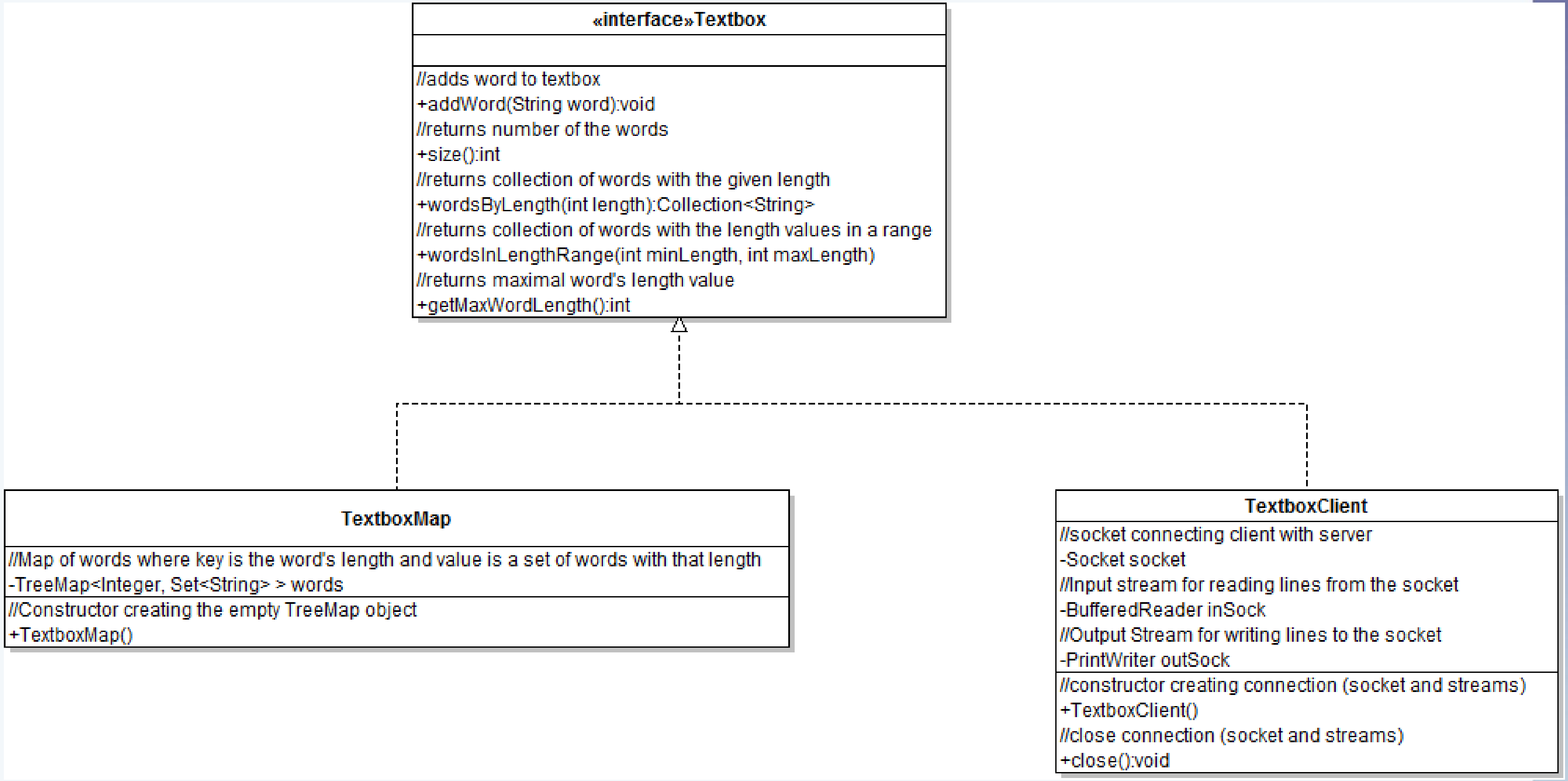
- Client gets two numbers (either integer or fractional) and operation from the console
- Client sends the numbers and operation to server
- Server gets the numbers and operation from the client
- Server computes and sends result back to client
- Client prints out the result on the console
- Optional requirement
  - In the calculator protocol the server shouldn’t use conditional statements

Exercise – Textbox Application

- Develop model according with UML diagram on the page 153
- Develop server application
  - Creates object of the class TextboxMap
  - Listens to a client and calls the proper methods of the TextboxMap class with returning result to the client
- Develop client application
  - Creates object of the class TextboxClient
  - Reads words from ***Dropbox\Java-6\Data\WarPeace.txt*** and using Textbox interface method performs the following
    - Parsing of words from line using split (“[^a-zA-Z]”)
    - Prints out all the longest words (words with the maximal length)
    - Prints out all words containing 10 letters (15 words on a line)
    - Prints out histogram (lines containing asterisks for all word length values)
      - Asterisks number on each line defines the percent's value of the amount of the words, length of which equals to the line’s number. If percent less than 1% one asterisk should be printed anyway
        - » Presentation example below of one line containing 18 asterisks shows that 18% from whole words are the words with length of 7 letters

        ```
        7 ******************
        ```

UML – Textbox Model


«interface»Textbox

//adds word to textbox
+addWord(String word):void
//returns number of the words
+size():int
//returns collection of words with the given length
+wordsByLength(int length):Collection<String>
//returns collection of words with the length values in a range
+wordsInLengthRange(int minLength, int maxLength)
//returns maximal word's length value
+getMaxWordLength():int

TextboxMap

//Map of words where key is the word's length and value is a set of words with that length
-TreeMap<Integer, Set<String> > words
//Constructor creating the empty TreeMap object
+TextboxMap()

TextboxClient

//socket connecting client with server
-Socket socket
//Input stream for reading lines from the socket
-BufferedReader inSock
//Output Stream for writing lines to the socket
-PrintWriter outSock
//constructor creating connection (socket and streams)
+TextboxClient()
//close connection (socket and streams)
+close():void

TEL-RAN