

OOP Summary

Francesco Ranellucci

dicembre 16, 2021

Contents

1	Objects	4
2	Classes	4
3	Constructor	4
4	Getters and Setters	5
5	ToString	5
6	General syntax	5
6.1	Array	5
6.2	if	6
6.3	for	6
6.3.1	Iterator Collection	6
6.4	while	6
6.5	do-while	6
7	Inheritance	6
7.1	Extends	6
7.2	Visibility	7
7.3	Super and This	7
8	polymorphism	8
8.1	Casting	9
8.1.1	Upcast	9
8.1.2	Downcast	9
8.2	Abstract class	10
8.3	Abstract modifier	10
8.4	Interfaces	10
9	Generic Class	10
9.1	Generic List	11
10	Useful Functions	11
10.1	compareTo	11
10.2	sort	11
10.3	11
11	Collection	11
11.1	List	14
11.1.1	LinkedList	15
11.1.2	ArrayList	16
11.1.3	Queue	17

11.1.4	Set	18
11.1.5	Delete	19
11.1.6	Add	20
11.2	Map	20
11.2.1	SortedMap	20
11.2.2	HashMap	20
11.2.3	TreeMap	21
12	Algorithms	21
12.1	Compare	21
12.2	Sort	21
12.3	Search	21
13	Exception	21
14	I/O files	24
14.1	Read a char	24
14.2	Read a char	24
14.3	Copying a text file	24
14.4	Copying a text file with buffer	25
15	Stream	25
15.0.1	Example	26
16	Exam Examples	27
16.1	Palestra	27
16.1.1	ExampleApp.java	27
16.1.2	Palestra.java	29
16.1.3	Inscritto.java	34
16.1.4	Esercizio.java	36
16.1.5	Weightlifting.java	36
16.1.6	Cardio.java	37
16.1.7	CorpoLibero.java	37
16.1.8	SchedaAllenamento.java	38
16.1.9	SchedaNonEsistenteException.java	39
16.1.10	UtenteNonEsistenteException.java	39
16.1.11	ComparatoreEsercizioCodice.java	39
16.1.12	ComparatoreCorpoLiberoCalorie.java	39
16.1.13	ComparatoreIscrittoNomeCognome.java	39
16.1.14	ComparatoreWeightliftingCaricoDescrescente.java	40

LICENSE

Notes taken for the course Objects-Oriented Programming Copyright © 2021 Francesco Ranellucci

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1 Objects

```
Vector v1 = new Vector();  
Vector v2 = new Vector();  
v1.sort();  
v1.search(22);
```

2 Classes

```
public class Car {  
  
    //attributes  
  
    String color;  
    String brand;  
    boolean turnedOn;  
  
    //methods  
  
    void turnOn() {  
        turnedOn = true;  
    }  
    void paint (String newCol) {  
        color = newCol;  
        printState  
    }  
    void printState () {  
        System.out.println(Car + brand + color);  
        System.out.println(the engine is +(turnedOn? on : off));  
    }  
}
```

3 Constructor

```
class Car {  
  
    // Default constructor, creates a red Ferrari  
  
    public Car(){  
        color = "red";  
        brand = "Ferrari";  
    }  
  
    // Constructor accepting the brand only  
  
    public Car(String carBrand){  
        color = "white";  
        brand = carBrand;  
    }  
  
    // Constructor accepting the brand and the color  
  
    public Car(String carBrand, String carColor){  
        color = carColor;  
        brand = carBrand;  
    }  
}
```

```

}
//=====

class Automobile {

    private String targa = new String();
    private String modello = new String();
    private int posto_assegnato;
    private int numero_giorni;

    public Automobile(String t, String mm, int p, int ng) {
        this.targa = t;
        this.modello = mm;
        this.posto_assegnato = p;
        this.numero_giorni = ng;
    }
}

```

4 Getters and Setters

```

{
    public String getTarga() {
        return targa;
    }
    public void setTarga(String targa) {
        this.targa = targa;
    }
    public String getModello() {
        return modello;
    }
    public void setModello(String modello) {
        this.modello = modello;
    }
}

```

5 ToString

```

{
    @Override
    public String toString() {
        return "Esercizio [codice=" + codice + ", descrizione=" + descrizione + "]";
    }
}

```

6 General syntax

6.1 Array

```

{
    int a[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
    int aa[] = new int [100];

    aa[0] = 3;
    int x = aa[1];
}

```

```

System.out.println("print array");
for (int i=0; i<a.length; i++){
    System.out.println(a[i]);
}
System.out.println("done");

//string

String stringhe[] = new String[10];
stringhe[0]= "Primo";
stringhe[1]= "Secondo";
stringhe[2]= "Terzo";
...

for (int i=0; i<stringhe.length; i++){
    System.out.println(stringhe[i]);
}
}

```

6.2 if

6.3 for

6.3.1 Iterator Collection

```

Collection<Person> persons = new LinkedList<Person>();

for(Iterator<Person> i = persons.iterator(); i.hasNext(); ) {
    Person p = i.next();
    System.out.println(p);
}

Collection persons = new LinkedList();

for(Iterator i= persons.iterator(); i.hasNext(); ) {
    Person p = (Person)i.next();
}

Collection<Person> persons = new LinkedList<Person>();

for(Person p: persons) {
    System.out.println(p);
}

```

6.4 while

6.5 do-while

7 Inheritance

7.1 Extends

```

{
    class Employee{
        String name;
        double wage;
        void incrementWage(){...}
    }
    class Manager extends Employee{

```

```

    String managedUnit;
    void changeUnit(){...}
}
Manager m = new Manager();
m.incrementWage(); // OK, inherited

class Employee{
    private String name;
    public void print(){
        System.out.println(name);
    }
}
class Manager extends Employee{
    private String managedUnit;
    public void print(){ //overrides that in Employee
        System.out.println(name); //un-optimized!
        System.out.println(managedUnit);
    }
}

Employee e1 = new Employee();
Employee e2 = new Manager();
e1.print();
e2.print();
}

```

7.2 Visibility

	Method in the same class	Method of another class in the same package	Method of subclass	Method of class in another package
private	✓			
package	✓	✓		
protected	✓	✓	✓	
public	✓	✓	✓	✓

Figure 1: image

7.3 Super and This

- this is a reference to the current object
- super is a reference to the parent class

```

class Car {

    String color;
    boolean isOn;
}

```

```

String licencePlate;

void paint(String color) {
    this.color = color;
}

void turnOn() {
    isOn=true;
}
}

class ElectricCar extends Car{
    boolean cellsAreCharged;

    void recharge() {
        cellsAreCharged = true;
    }

    void turnOn() {
        if( cellsAreCharged )
            super.turnOn();
    }
}

class Employee {
    private String name;
    private double wage;
    Employee(String n, double w){
        name = n;
        wage = w;
    }
}

class Manager extends Employee {
    private int unit;
    Manager(String n, double w, int u) {
        super(n,w); // ok
        unit = u;
    }
}

```

8 polymorphism

- Polymorphism: allows feeding algorithms with different objects
- Dynamic binding: allows accommodating different behavior behind the same interface

```

Car myCar;
myCar = new Car();
myCar = new ElectricCar();

Car[] garage = new Car[4];
garage[0] = new Car();
garage[1] = new ElectricCar();
garage[2] = new ElectricCar();
garage[3] = new Car();

for(int i=0; i<garage.length; i++){

```



```

    garage[i].turnOn();
}

for(Car a : garage){
    a.turnOn();
}

```

References of type Object play a role similar to void* in C

```

Object [] objects = new Object[3];
objects[0] = "First!";
objects[2] = new Employee("Luca","Verdi");
objects[1] = new Integer(2);
for(Object obj : objects){
    System.out.println(obj);
}

```

8.1 Casting

```

float f;
f = 4.7; // legal
f = "string"; // illegal
Car c;
c = new Car(); // legal
c = new String(); // illegal

class Car{};
class ElectricCar extends Car{};
Car c = new Car();
ElectricCar ec = new ElectricCar ();

class Car{};
class ElectricCar extends Car{};
Car a = new ElectricCar ();

```

8.1.1 Upcast

```

Car c = new Car();
ElectricCar ec = new ElectricCar();
c = ec;

```

8.1.1.1 Upcast to Object

```

AnyClass foo = new AnyClass();
Object obj;
obj = foo;

```

8.1.2 Downcast

```

Car c = new ElectricCar(); // implic. upcast
c.recharge(); // wrong!
// explicit downcast
ElectricCar ec = (ElectricCar)c;
ec.recharge(); // ok

Car c = new Car();
ElectricCar ec;
if (c instanceof ElectricCar ){
    ec = (ElectricCar) c;
}

```

```
ec.recharge();
}
```

8.2 Abstract class

8.3 Abstract modifier

8.4 Interfaces

Interface `implements` Car

Class `implements` Car

Class `implements` Comparable<Car>

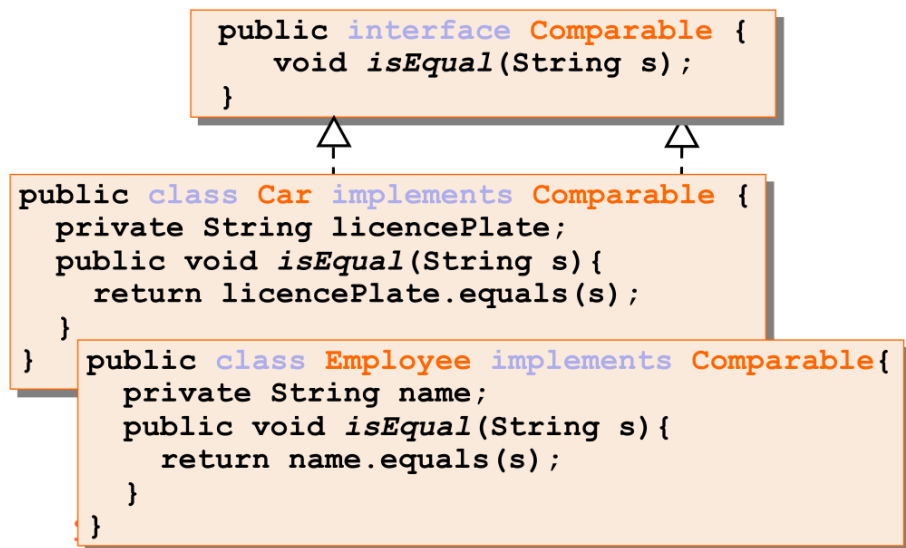


Figure 2: image

9 Generic Class

```
public class Person<T> {
    String first;
    String last;
    T ID;
    Person(String first,String last,T ID){
        this.first = first;
        this.last = last;
        this.ID = ID;
    }
    T getID(){ return ID; }
}
```

```
Person<Integer> a = new Person<Integer> ("A1","A",new Integer(123));
Person<String> b = new Person<String> ("Pat","B","s32");
```

```
Integer id1 = a.getID();
String id2 = b.getID();
Integer ids = b.getID();
```

9.1 Generic List

```
public interface List<E>{  
    void add(E x);  
    Iterator<E> iterator();  
}  
  
public interface Iterator<E>{  
    E next();  
    boolean hasNext();  
}
```

10 Useful Functions

10.1 compareTo

10.2 sort

10.3

11 Collection

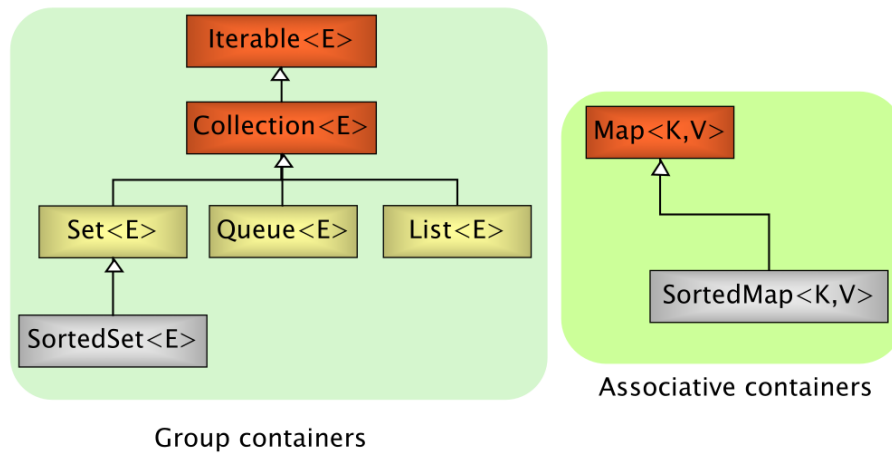


Figure 3: image

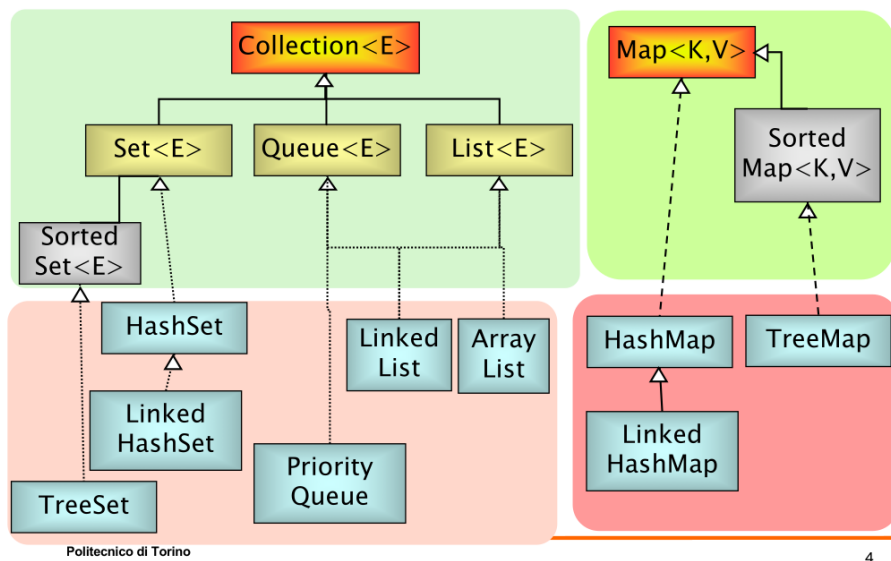


Figure 4: image

data structure

	Hash table	Resizable array	Balanced tree	Linked list	Hash table Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

interface

classes

Figure 5: image

Collection interface

- `int size()`
- `boolean isEmpty()`
- `boolean contains(E element)`
- `boolean containsAll(Collection<?> c)`
- `boolean add(E element)`
- `boolean addAll(Collection<? extends E> c)`
- `boolean remove(E element)`
- `boolean removeAll(Collection<?> c)`
- `void clear()`
- `Object[] toArray()`
- `Iterator<E> iterator()`

Figure 6: image

```
Collection<Person> persons = new LinkedList<Person>();
```

```
persons.add( new Person("Alice") );
```

```
System.out.println( persons.size() );
```

```
Collection<Person> copy = new TreeSet<Person>();
```

```
copy.addAll(persons); // new TreeSet(persons)
```

```
Person[] array = copy.toArray();
```

```
System.out.println( array[0] );
```

List interface: further methods

- `E get(int index)`
- `E set(int index, E element)`
- `void add(int index, E element)`
- `E remove(int index)`

- `boolean addAll(int index, Collection<E> c)`
- `int indexOf(E o)`
- `int lastIndexOf(E o)`
- `List<E> subList(int from, int to)`

Figure 7: image

ArrayList

- `get(n)`
 - ♦ Constant
- Insert/add (beginning) and delete while iterating
 - ♦ Linear

LinkedList

- `get(n)`
 - ♦ Linear
- Insert/add (beginning) and delete while iterating
 - ♦ Constant

Figure 8: image

11.1.1.1 LinkedList

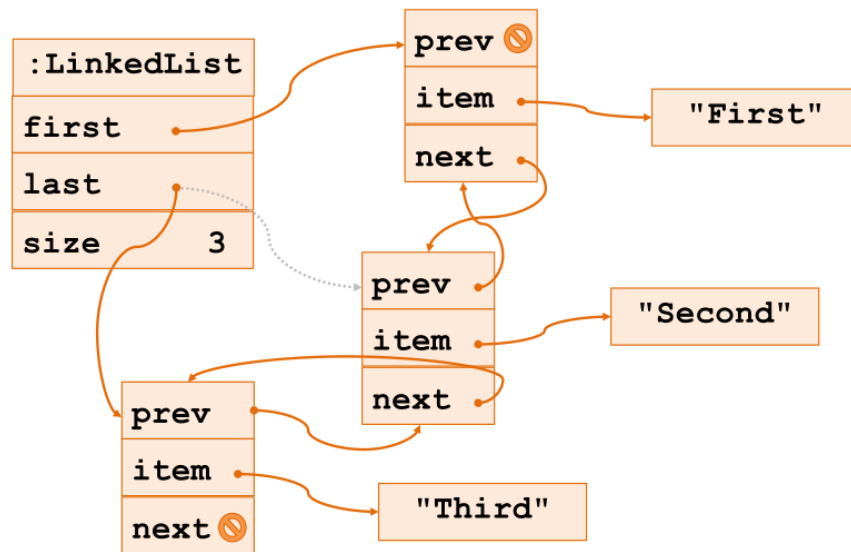


Figure 9: image

```
LinkedList<Integer> ll = new LinkedList<Integer>();
```

```
ll.add(new Integer(10));
```

```
ll.add(new Integer(11));
```

```
ll.addLast(new Integer(13));
```

```
ll.addFirst(new Integer(20));
```

```
List<Car> garage = new ArrayList<Car>(20);
```

```
garage.set( 0, new Car() );
```

```
garage.set( 1, new ElectricCar() );
```

```
garage.set( 2, new ElectricCar() );
```

```
garage.set( 3, new Car());
```

```
for(int i; i<garage.size(); i++){
```

```
    Car c = garage.get(i);
```

```
    c.turnOn();
```

```
}
```

```
List l = new ArrayList(2); // 2 refs to null
```

```
l.add(new Integer(11));    // 11 in position 0
```

```
l.add(0, new Integer(13)); // 11 in position 1
```

```
l.set(0, new Integer(20)); // 13 replaced by 20
```

```
l.add(9, new Integer(30)); // NO: out of bounds
```

```
l.add(new Integer(30));    // OK, size extended
```

11.1.2 ArrayList

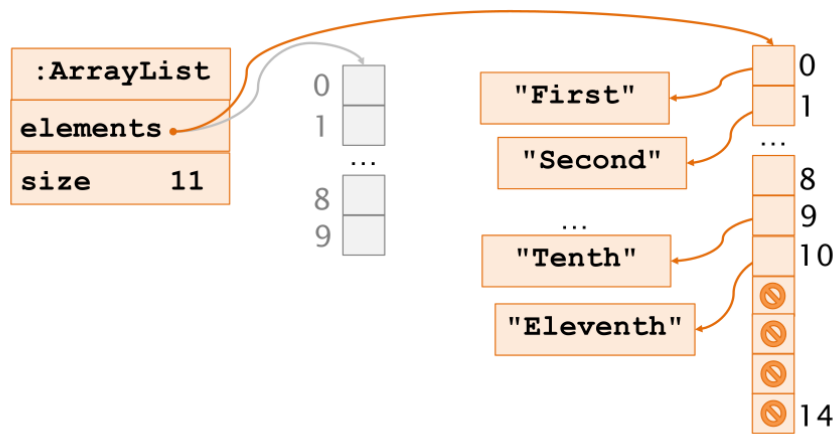


Figure 10: image

11.1.3 Queue

Queue implementations - LinkedList - Head is the first element of the list - FIFO: First-In-First-Out - PriorityQueue
- Head is the smallest element

```
Queue<Integer> fifo = new LinkedList<Integer>();  
Queue<Integer> pq = new PriorityQueue<Integer>();
```

```
fifo.add(3); pq.add(3);  
fifo.add(1); pq.add(1);  
fifo.add(2); pq.add(2);
```

```
System.out.println(fifo.peek()); // 3  
System.out.println(pq.peek()); // 1
```

Set implementations

- **HashSet** implements **Set**
 - ♦ Hash tables as internal data structure (faster)
- **LinkedHashSet** extends **HashSet**
 - ♦ Elements are traversed according to the **insertion order**
- **TreeSet** implements **SortedSet**
 - ♦ R-B trees as internal data structure (computationally expensive)

Figure 11: image

11.1.5 Delete

```
List<Integer> lst=new LinkedList<Integer>();

lst.add(new Integer(10));
lst.add(new Integer(11));
lst.add(new Integer(13));
lst.add(new Integer(20));

int count = 0;
for (Iterator<?> itr = lst.iterator(); itr.hasNext(); ) {
    itr.next();
    if (count==1)
        itr.remove(); // ok
    count++;
}
```

11.1.6 Add

```
List lst = new LinkedList();

lst.add(new Integer(10));
lst.add(new Integer(11));
lst.add(new Integer(13));
lst.add(new Integer(20));

int count = 0;
for (Iterator itr = lst.iterator(); itr.hasNext(); ) {
    itr.next();
    if (count==2)
        itr.add(new Integer(22)); // ok
    count++;
}
```

11.2 Map

```
Map<String,Person> people = new HashMap<String,Person>();

people.put( "ALCSMT", /*ssn*/ new Person("Alice", "Smith") );
people.put( "RBTGRN", /*ssn*/ new Person("Robert", "Green") );

Person bob = people.get("RBTGRN");
if( bob == null )
    System.out.println( "Not found" );
int populationSize = people.size();
```

11.2.1 SortedMap

11.2.2 HashMap

```
Map<String,Student> students = new HashMap<String,Student>();

students.put("123", new Student("123","Joe Smith"));

Student s = students.get("123");

for(Student si: students.values()){

}
```

11.2.2.1 Iteration

```
Map<String,Person> people = new HashMap<String,Person>();

Collection<Person> values = people.values();

for(Person p: values) {
    System.out.println(p);
}
```

11.2.2.2 Print all key

```
Map<String,Person> people = new HashMap<String,Person>();

Collection<String> keys = people.keySet();
for(String ssn: keys) {
```

```

    Person p = people.get(ssn);
    System.out.println(ssn + " - " + p);
}

```

11.2.3 TreeMap

12 Algorithms

Algorithms

- Static methods of java.util.Collections class
- **Work on lists**, since it has the concept of position
 - ♦ **sort()** – merge sort, $n \log(n)$
 - ♦ **binarySearch()** – requires ordered sequence
 - ♦ **shuffle()** – unsort
 - ♦ **reverse()** – requires ordered sequence
 - ♦ **rotate()** – of given a distance
 - ♦ **min()** , **max()** – in a Collection

Figure 12: image

12.1 Compare

```

class StudentIDComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2){
        return s1.getID() - s2.getID();
    }
}

```

12.2 Sort

```

List students = new LinkedList();

students.add(new Student("Mary","Smith",34621));
students.add(new Student("Alice","Knight",13985));
students.add(new Student("Joe","Smith",95635));

Collections.sort(students); // sort by name
Collections.sort(students, new StudentIDComparator()); // sort by ID

```

12.3 Search

Binary search

13 Exception

- Java provides three keywords

```
try {
    open the file;
    determine file size;
    allocate that much memory;
    read the file into memory;
    close the file;
} catch (fileOpenFailed) {
    doSomething;
} catch (sizeDeterminationFailed) {
    doSomething;
} catch (memoryAllocationFailed) {
    doSomething;
} catch (readFailed) {
    doSomething;
} catch (fileCloseFailed) {
    doSomething;
}
```

Figure 13: image

- Throw
 - * Raises (generate) an exception
 - Try
 - * Introduces code to watch for exceptions
 - Catch
 - * Defines the exception handling code
 - Java also defines a new type
 - * Throwable (and Exception)
1. Identify/Define an exception class
 2. Declare/Mark the method as potential source of exception
 3. Create an exception object
 4. Throw upward the exception

```
// java.lang.Exception
public class EmptyStack extends Exception {
}
class Stack<E>{
    public E pop() throws EmptyStack {

        if(size == 0) {
            Exception e = new EmptyStack();
            throw e;
        }
    }
}
```

```
try {
    // in this piece of code some
    // exceptions may be generated
    stack.pop();
    ...
}
catch (StackEmpty e) {
    // error handling
    System.out.println(e);
    ...
}
```

```
class Dummy {
    public void foo() throws FileNotFoundException{
        FileReader f;
        f = new FileReader("file.txt");
    }
}
```

```
class Dummy {
    public void foo() throws FileNotFoundException {
        try{
            FileReader f;
            f = new FileReader("file.txt");
        } catch (FileNotFoundException fnf) {
```

```

        // handle fnf, e.g., print it
        throw fnf;
    }
}
}

```

14 I/O files

14.1 Read a char

```

int ch = r.read();
char unicode = (char) ch;

System.out.print(unicode);
r.close();

```

14.2 Read a char

```

public static String readLine(Reader r) throws IOException{

    StringBuffer res= new StringBuffer();
    int ch = r.read();
    if(ch == -1) return null; // END OF FILE!
    while( ch != -1 ){
        char unicode = (char) ch;
        if(unicode == '\n') break;
        if(unicode != '\r')
            res.append(unicode);
        ch = r.read();
    }
    return res.toString();
}

```

14.3 Copying a text file

```

import java.io.*;
public class Copy {
    public static void main(String[] args) throws IOException{
        File inputFile = new File("in.txt");
        File outputFile = new File("out.txt");
        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;
        while ((c = in.read()) != -1)
            out.write(c); // One char at a time, inefficient
        in.close();
        out.close();
    }
}

```

```

import java.io.*;
public class Copy {
    public static void main(String[] args) throws
        IOException{
        FileReader in = new FileReader("in.txt");
        FileWriter out = new FileWriter("out.txt");

```



```

        int c;
        while ((c = in.read()) != -1)
            out.write(c); // One char at a time, inefficient
        in.close();
        out.close();
    }
}

```

14.4 Copying a text file with buffer

```

import java.io.*;
public class Copy {
    public static void main(String[] args) throws
        IOException{
        FileReader in = new FileReader("in.txt");
        FileWriter out = new FileWriter("out.txt");
        char[] buffer = new char[4096];
        int n;
        while ((n = in.read(buffer)) != -1)
            out.write(buffer, 0, n);
        in.close();
        out.close();
    }
}

```

15 Stream

- Arrays
 - Stream stream()

```
String[] s={"Red", "Green", "Blue"}.Arrays.stream(s).forEach(System.out::println)
```
- Stream of
 - static Stream of(T... values)

```
Stream.of("Red", "Green", "Blue").forEach(System.out::println);
```
- Collection
 - Stream stream()

```
Collection<Student> oopClass = new LinkedList<>();

oopClass.add(new Student(100,"John","Smith"));
oopClass.stream().forEach(System.out::println);
```

```

//map
List number = Arrays.asList(2,3,4,5);
List square = number.stream().map(x->x*x).collect(Collectors.toList());

//filter
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());

//sorted
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().sorted().collect(Collectors.toList());

//collect
List number = Arrays.asList(2,3,4,5,3);
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());

```

```

//foreach
List number = Arrays.asList(2,3,4,5);
number.stream().map(x->x*x).forEach(y->System.out.println(y));

//reduce
List number = Arrays.asList(2,3,4,5);
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

```

15.0.1 Example

```

//a simple program to demonstrate the use of stream in java
import java.util.*;
import java.util.stream.*;

class Demo {
    public static void main(String args[]) {

        // create a list of integers
        List<Integer> number = Arrays.asList(2,3,4,5);

        // demonstration of map method
        List<Integer> square = number.stream().map(x -> x*x).collect(Collectors.toList());
        System.out.println(square);

        // create a list of String
        List<String> names = Arrays.asList("Reflection","Collection","Stream");

        // demonstration of filter method
        List<String> result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());
        System.out.println(result);

        // demonstration of sorted method
        List<String> show = names.stream().sorted().collect(Collectors.toList());
        System.out.println(show);

        // create a list of integers
        List<Integer> numbers = Arrays.asList(2,3,4,5,2);

        // collect method returns a set
        Set<Integer> squareSet = numbers.stream().map(x->x*x).collect(Collectors.toSet());
        System.out.println(squareSet);

        // demonstration of forEach method
        number.stream().map(x->x*x).forEach(y->System.out.println(y));

        // demonstration of reduce method
        int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

        System.out.println(even);
    }
}

```

16 Exam Examples

16.1 Palestra

16.1.1 ExampleApp.java

```
import java.util.ArrayList;
import palestra.*;

public class Esempio {

    public static void main(String[] args) throws SchedaNonEsistenteException, UtenteNonEsistenteException {

        Palestra p = new Palestra();

        System.out.println("\n/***** R1 *****/");

        System.out.println("Nuove iscrizioni");
        Iscritto i1 = p.nuovaIscrizione("Mario", "Rossi", "Uomo", 25, 84.5);
        Iscritto i2 = p.nuovaIscrizione("Paolo", "Bianchi", "Uomo", 28, 80.1);
        Iscritto i3 = p.nuovaIscrizione("Anna", "Verdi", "Donna", 22, 57.9);

        System.out.println("\nUtenti iscritti:\n");
        System.out.println(i1.descriviti());
        System.out.println(i2.descriviti());
        System.out.println(i3.descriviti());

        System.out.println("\nRicerca iscritto:\n");
        Iscritto iTrovato = p.cercaIscrittoPerId(1);
        System.out.println(iTrovato.descriviti());

        System.out.println("\nRicerca iscritto/i per nome e cognome:\n");
        ArrayList<Iscritto> iTrovatiNomeCognome = new ArrayList<>(p.cercaIscrittoPerNomeCognome("o", "i"))
        for(Iscritto ii : iTrovatiNomeCognome)
            System.out.println(ii.descriviti());

        System.out.println("\nElenco iscritti:\n");
        ArrayList<Iscritto> elencoIscritti = new ArrayList<>(p.elencoIscritti());
        for(Iscritto ii : elencoIscritti)
            System.out.println(ii.descriviti());

        System.out.println("\n/***** R2 *****/");

        System.out.println("Nuovi esercizi");
        Esercizio e1 = p.nuovoEsercizio("rpj", "Rope-Jump", 20);
        Esercizio e2 = p.nuovoEsercizio("sqt", "Squat", 6, 120);
        Esercizio e3 = p.nuovoEsercizio("brp", "Burpees", 10);
        Esercizio e4 = p.nuovoEsercizio("psu", "Push-up", 24.5);
        Esercizio e5 = p.nuovoEsercizio("pu", "Pull-up", 34.5);
        Esercizio e6 = p.nuovoEsercizio("ddl", "Deadlift", 6, 150);

        System.out.println("\nEsercizi creati:\n");
        System.out.println(e1.descriviti());
        System.out.println(e2.descriviti());
        System.out.println(e3.descriviti());
        System.out.println(e4.descriviti());
        System.out.println(e5.descriviti());
```

```

System.out.println(e6.descriviti());

System.out.println("\nRicerca esercizio:\n");
Esercizio eTrovato = p.esercizio("sqt");
System.out.println(eTrovato.descriviti());

System.out.println("\nElenco esercizi:\n");
ArrayList<Esercizio> elencoEsercizi = new ArrayList<>(p.esercizi());
for(Esercizio ei : elencoEsercizi)
    System.out.println(ei.descriviti());

System.out.println("\nElenchi esercizi");
System.out.println("\nCodice, alfabeticamente:\n");
ArrayList<Esercizio> elenco1 = new ArrayList<>(p.elencoEserciziPerCodice());
for(Esercizio ei : elenco1)
    System.out.println(ei.descriviti());

System.out.println("\nTipologia:\n");
ArrayList<Esercizio> elenco2 = new ArrayList<>(p.elencoEserciziPerTipologia());
for(Esercizio ei : elenco2)
    System.out.println(ei.descriviti());

System.out.println("\nCorpo libero per calorie:\n");
ArrayList<Esercizio> elenco3 = new ArrayList<>(p.elencoEserciziCorpoLiberoPerCalorie());
for(Esercizio ei : elenco3)
    System.out.println(ei.descriviti());

System.out.println("\n/***** R3 *****/");

System.out.println("Nuove schede");
ArrayList<String> eserciziScheda1 = new ArrayList<>();
eserciziScheda1.add("psu");
eserciziScheda1.add("rpj");
SchedaAllenamento s1 = p.nuovaSchedaAllenamento(0, "2021/12/03", eserciziScheda1);

System.out.println("\nScheda creata:\n");
for(Esercizio ei : p.eserciziScheda(s1.getCodice()))
    System.out.println(ei.descriviti());

System.out.println("\nAggiunta esercizi:\n");
eserciziScheda1.add("sqt");
p.nuovaSchedaAllenamento(0, "2021/12/03", eserciziScheda1);
for(Esercizio ei : p.eserciziScheda("2021/12/03_0"))
    System.out.println(ei.descriviti());

System.out.println("\nRicerca scheda:\n");
SchedaAllenamento sTrovata = p.cercaSchedaPerId("2021/12/03_0");
for(Esercizio ei : p.eserciziScheda(sTrovata.getCodice()))
    System.out.println(ei.descriviti());

System.out.println("\nElenco schede per iscritto\n");
ArrayList<SchedaAllenamento> schedeTrovate = new ArrayList<>(p.elencoSchedePerIdIscritto(0));

for (SchedaAllenamento si : schedeTrovate) {
    System.out.println("Scheda:" + si.getCodice());
    for(Esercizio ei : p.eserciziScheda(si.getCodice()))

```

```

        System.out.println(ei.descriviti());
    }

    System.out.println("\n/***** R4 *****/");

    System.out.println("\nLettura da file:\n");
    Palestra p2 = new Palestra();

    p2.leggiDatiPalestra("input.txt");

    System.out.println("\nElenco iscritti:\n");
    ArrayList<Iscritto> elencoIscritti2 = new ArrayList<>(p2.elencoIscritti());
    for(Iscritto ii : elencoIscritti2)
        System.out.println(ii.descriviti());

    System.out.println("\nElenco esercizi:\n");
    ArrayList<Esercizio> elencoEsercizi2 = new ArrayList<>(p2.esercizi());
    for(Esercizio ei : elencoEsercizi2)
        System.out.println(ei.descriviti());
    }
}

```

16.1.2 Palestra.java

```

package palestra;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.LinkedHashMap;

public class Palestra {

    LinkedHashMap<Integer, Iscritto> iscritti;
    LinkedHashMap<String, Esercizio> esercizi;
    LinkedHashMap<String, SchedaAllenamento> schede;

    public Palestra() {
        iscritti = new LinkedHashMap<>();
        esercizi = new LinkedHashMap<>();
        schede = new LinkedHashMap<>();
    }

    public Iscritto nuovaIscrizione(String nome, String cognome, String sesso, int eta, double peso) {

        Iscritto i = null;
        int codice = iscritti.size();
        i = new Iscritto(codice, nome, cognome, sesso, eta, peso);
        iscritti.put(codice, i);

        return i;
    }
}

```

```

public Iscritto cercaIscrittoPerId(int codice) {
    return iscritti.get(codice);
}

public Collection<Iscritto> cercaIscrittoPerNomeCognome(String nome, String cognome){

    ArrayList<Iscritto> ii = new ArrayList<>();

    for (Iscritto i : iscritti.values()) {
        if (i.getName().contains(nome) && i.getCognome().contains(cognome))
            ii.add(i);
    }

    Collections.sort(ii, new ComparatoreIscrittoNomeCognome());

    return ii;
}

public Collection<Iscritto> elencoIscritti(){
    ArrayList<Iscritto> ii = new ArrayList<>(iscritti.values());

    Collections.sort(ii, new ComparatoreIscrittoNomeCognome());

    return ii;
}

public Cardio nuovoEsercizio(String codice, String descrizione, int minuti) {
    Cardio c = null;

    if (!esercizi.containsKey(codice)) {
        c = new Cardio(codice, descrizione, minuti);
        esercizi.put(codice, c);
    }

    return c;
}

public Weightlifting nuovoEsercizio(String codice, String descrizione, int ripetizioni, int carico) {
    Weightlifting w = null;

    if (!esercizi.containsKey(codice)) {
        w = new Weightlifting(codice, descrizione, ripetizioni, carico);
        esercizi.put(codice, w);
    }

    return w;
}

public CorpoLibero nuovoEsercizio(String codice, String descrizione, double calorie) {
    CorpoLibero c = null;

    if (!esercizi.containsKey(codice)) {
        c = new CorpoLibero(codice, descrizione, calorie);
        esercizi.put(codice, c);
    }
}

```

```

        return c;
    }

    public Esercizio esercizio(String codice) {
        return esercizi.get(codice);
    }

    public Collection<Esercizio> esercizi() {

        if (esercizi.size() == 0)
            return null;

        return esercizi.values();
    }

    public Collection<Esercizio> elencoEserciziPerCodice() {

        ArrayList<Esercizio> ee = new ArrayList<>(esercizi.values());
        Collections.sort(ee, new ComparatoreEsercizioCodice());

        return ee;
    }

    public Collection<Esercizio> elencoEserciziPerTipologia() {

        ArrayList<Esercizio> res = new ArrayList<>();
        ArrayList<Esercizio> car = new ArrayList<>();
        ArrayList<Esercizio> wgt = new ArrayList<>();
        ArrayList<Esercizio> cpl = new ArrayList<>();

        for (Esercizio ei : esercizi.values()) {
            if (ei instanceof Cardio)
                car.add(ei);
            if (ei instanceof Weightlifting)
                wgt.add(ei);
            if (ei instanceof CorpoLibero)
                cpl.add(ei);
        }

        Collections.sort(wgt, new ComparatoreWeightliftingCaricoDescrescente());
        res.addAll(car);
        res.addAll(wgt);
        res.addAll(cpl);

        return res;
    }

    public Collection<Esercizio> elencoEserciziCorpoLiberoPerCalorie() {

        ArrayList<Esercizio> cpl = new ArrayList<>();

        for (Esercizio ei : esercizi.values()) {
            if (ei instanceof CorpoLibero)
                cpl.add(ei);
        }
    }

```

```

        Collections.sort(cpl, new ComparatoreCorpoLiberoCalorie());

        return cpl;
    }

    public SchedaAllenamento nuovaSchedaAllenamento(int codiceIscritto, String data,
        Collection<String> codiciEsercizi) {

        ArrayList<Esercizio> eserciziEsistenti = new ArrayList<>();
        Iscritto i = cercaIscrittoPerId(codiceIscritto);
        SchedaAllenamento sa = null;

        for (String s : codiciEsercizi) {
            Esercizio e = esercizio(s);
            if (e != null) {
                eserciziEsistenti.add(e);
            }
        }

        if (i != null && eserciziEsistenti.size() > 0) {

            String codice = data + "_" + i.getCodice();

            if (schede.containsKey(codice)) {
                sa = schede.get(codice);
                sa.aggiungiEsercizi(eserciziEsistenti);
            }
            else {
                sa = new SchedaAllenamento(codice, i, eserciziEsistenti);
                schede.put(codice, sa);
                i.aggiungiScheda(sa);
            }
        }

        return sa;
    }

    public Collection<Esercizio> eserciziScheda(String codiceScheda){

        ArrayList<Esercizio> ee = null;
        SchedaAllenamento sa = schede.get(codiceScheda);
        if (sa != null)
            ee = new ArrayList<>(sa.getEsercizi());

        return ee;
    }

    public SchedaAllenamento cercaSchedaPerId(String codiceScheda) throws SchedaNonEsistenteException{

        SchedaAllenamento sa = schede.get(codiceScheda);
        if (sa==null) {
            throw new SchedaNonEsistenteException();
        }

        return sa;
    }

```



```

    }

    public Collection<SchedaAllenamento> elencoSchedePerIdIscritto(int codiceIscritto)
        throws UtenteNonEsistenteException{
        Iscritto i = cercaIscrittoPerId(codiceIscritto);
        ArrayList<SchedaAllenamento> ss = null;

        if (i == null)
            throw new UtenteNonEsistenteException();

        else {
            ss = new ArrayList<>(i.getSchede());
        }

        return ss;
    }

    public void leggiDatiPalestra(String nomeFile) {

        FileReader f;
        try {
            f = new FileReader(nomeFile);
            try (BufferedReader b = new BufferedReader(f)) {
                String line = "";

                while ((line = b.readLine()) != null){
                    String[] fields = line.split(";");
                    try {
                        if (fields[0].compareTo("I")==0) {
                            nuovaIscrizione(fields[1], fields[2], fields[3], Integer.parseInt(fields[4]),
                                Double.parseDouble(fields[5]));
                        }
                    }
                    catch(IndexOutOfBoundsException e){
                        e.printStackTrace();
                    }
                    catch(NumberFormatException e){
                        e.printStackTrace();
                    }
                    try {
                        if (fields[0].compareTo("C")==0) {
                            nuovoEsercizio(fields[1], fields[2], Integer.parseInt(fields[3]));
                        }
                    }
                    catch(IndexOutOfBoundsException e){
                        e.printStackTrace();
                    }
                    catch(NumberFormatException e){
                        e.printStackTrace();
                    }
                    try {
                        if (fields[0].compareTo("W")==0) {
                            nuovoEsercizio(fields[1], fields[2], Integer.parseInt(fields[3]),
                                Integer.parseInt(fields[4]));
                        }
                    }
                }
            }
        }
    }

```

```

        catch(IndexOutOfBoundsException e){
            e.printStackTrace();
        }
        catch(NumberFormatException e){
            e.printStackTrace();
        }
        try {
            if (fields[0].compareTo("P")==0) {
                nuovoEsercizio(fields[1], fields[2], Double.parseDouble(fields[3]));
            }
        }
        catch(IndexOutOfBoundsException e){
            e.printStackTrace();
        }
        catch(NumberFormatException e){
            e.printStackTrace();
        }
    }
}
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}
}

```

16.1.3 Inscritto.java

```

package palestra;

import java.util.ArrayList;
import java.util.Collection;

public class Iscritto {

    public int getCodice() {
        return codice;
    }

    public void setCodice(int codice) {
        this.codice = codice;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCognome() {
        return cognome;
    }
}

```

```

public void setCognome(String cognome) {
    this.cognome = cognome;
}

public String getSesso() {
    return sesso;
}

public void setSesso(String sesso) {
    this.sesso = sesso;
}

public int getEta() {
    return eta;
}

public void setEta(int eta) {
    this.eta = eta;
}

public double getPeso() {
    return peso;
}

public void setPeso(double peso) {
    this.peso = peso;
}

private int codice;
private String nome;
private String cognome;
private String sesso;
private int eta;
private double peso;
ArrayList<SchedaAllenamento> schedi;

public Iscritto(int codice, String nome, String cognome, String sesso, int eta, double peso) {
    this.codice = codice;
    this.nome = nome;
    this.cognome = cognome;
    this.sesso = sesso;
    this.eta = eta;
    this.peso = peso;
    // TODO Auto-generated constructor stub
    schedi = new ArrayList<>();
}

public String descriviti() {
    return codice + " " + nome + " " + cognome + " " + sesso + " " + eta + " " + peso;
}

public void aggiungiScheda(SchedaAllenamento sa) {
    // TODO Auto-generated method stub
    schedi.add(sa);
}

```

```

    public Collection<SchedaAllenamento> getSchede(){
        return schede;
    }
}

```

16.1.4 Esercizio.java

```

package palestra;

public class Esercizio {

    public String getCodice() {
        return codice;
    }

    public void setCodice(String codice) {
        this.codice = codice;
    }

    public String getDescrizione() {
        return descrizione;
    }

    public void setDescrizione(String descrizione) {
        this.descrizione = descrizione;
    }

    private String codice;
    private String descrizione;

    public Esercizio(String codice, String descrizione) {
        this.codice = codice;
        this.descrizione = descrizione;
        // TODO Auto-generated constructor stub
    }

    public String descriviti() {
        return codice + " " + descrizione;
    }
}

```

16.1.5 Weightlifting.java

```

package palestra;

public class Weightlifting extends Esercizio {
    public int getRipetizioni() {
        return ripetizioni;
    }

    public void setRipetizioni(int ripetizioni) {
        this.ripetizioni = ripetizioni;
    }
}

```

```

    public int getCarico() {
        return carico;
    }

    public void setCarico(int carico) {
        this.carico = carico;
    }

    private int ripetizioni;
    private int carico;

    public Weightlifting(String codice, String descrizione, int ripetizioni, int carico) {
        // TODO Auto-generated constructor stub
        super(codice, descrizione);
        this.ripetizioni = ripetizioni;
        this.carico = carico;
    }

    public String descriviti() {
        return super.descriviti() + " WGT " + ripetizioni + " " + carico;
    }
}

```

16.1.6 Cardio.java

```

package palestra;

public class Cardio extends Esercizio {

    private int minuti;

    public Cardio(String codice, String descrizione, int minuti) {
        // TODO Auto-generated constructor stub
        super(codice, descrizione);
        this.minuti = minuti;
    }

    public String descriviti() {
        return super.descriviti() + " CAR " + minuti;
    }
}

```

16.1.7 CorpoLibero.java

```

package palestra;

public class CorpoLibero extends Esercizio {

    public double getCalorie() {
        return calorie;
    }

    public void setCalorie(double calorie) {
        this.calorie = calorie;
    }

    private double calorie;
}

```

```

public CorpoLibero(String codice, String descrizione, double calorie) {
    super(codice, descrizione);
    this.calorie = calorie;
    // TODO Auto-generated constructor stub
}

public String descriviti() {
    return super.descriviti() + " CPL " + calorie;
}

}

```

16.1.8 SchedaAllenamento.java

```

package palestra;

import java.util.ArrayList;
import java.util.Collection;

public class SchedaAllenamento {

    private String codice;
    private Iscritto i;
    private ArrayList<Esercizio> eserciziEsistenti;

    public SchedaAllenamento(String codice, Iscritto i, ArrayList<Esercizio> eserciziEsistenti) {
        this.codice = codice;
        this.setI(i);
        this.eserciziEsistenti = eserciziEsistenti;
        // TODO Auto-generated constructor stub
    }

    public String getCodice() {
        return codice;
    }

    public void aggiungiEsercizi(ArrayList<Esercizio> esercizi) {
        // TODO Auto-generated method stub
        for (Esercizio ei : esercizi) {
            if (!eserciziEsistenti.contains(ei))
                eserciziEsistenti.add(ei);
        }
    }

    public Collection<Esercizio> getEsercizi() {
        // TODO Auto-generated method stub
        return eserciziEsistenti;
    }

    public Iscritto getI() {
        return i;
    }

    public void setI(Iscritto i) {
        this.i = i;
    }
}

```

```

    }
}

```

16.1.9 SchedaNonEsistenteException.java

```

package palestra;

@SuppressWarnings("serial")
public class SchedaNonEsistenteException extends Exception {

}

```

16.1.10 UtenteNonEsistenteException.java

```

package palestra;

@SuppressWarnings("serial")
public class UtenteNonEsistenteException extends Exception {

}

```

16.1.11 ComparatoreEsercizioCodice.java

```

package palestra;

import java.util.Comparator;

public class ComparatoreEsercizioCodice implements Comparator<Esercizio> {

    @Override
    public int compare(Esercizio o1, Esercizio o2) {
        // TODO Auto-generated method stub
        return o1.getCodice().compareTo(o2.getCodice());
    }

}

```

16.1.12 ComparatoreCorpoLiberoCalorie.java

```

package palestra;

import java.util.Comparator;

public class ComparatoreCorpoLiberoCalorie implements Comparator<Esercizio> {

    @Override
    public int compare(Esercizio o1, Esercizio o2) {
        // TODO Auto-generated method stub
        CorpoLibero c1 = (CorpoLibero) o1;
        CorpoLibero c2 = (CorpoLibero) o2;
        return (int)(c1.getCalorie() - c2.getCalorie());
    }

}

```

16.1.13 ComparatoreIscrittoNomeCognome.java

```

package palestra;

```

```

import java.util.Comparator;

public class ComparatoreIscrittoNomeCognome implements Comparator<Iscritto> {

    @Override
    public int compare(Iscritto o1, Iscritto o2) {
        // TODO Auto-generated method stub

        String nc1 = o1.getNome()+o1.getCognome();
        String nc2 = o2.getNome()+o2.getCognome();

        return nc1.compareTo(nc2);
    }
}

```

16.1.14 ComparatoreWeightliftingCaricoDescrescente.java

```

package palestra;

import java.util.Comparator;

public class ComparatoreWeightliftingCaricoDescrescente implements Comparator<Esercizio> {

    @Override
    public int compare(Esercizio o1, Esercizio o2) {
        // TODO Auto-generated method stub
        Weightlifting w1 = (Weightlifting) o1;
        Weightlifting w2 = (Weightlifting) o2;
        return w2.getCarico()-w1.getCarico();
    }
}

```