1) Scrivi un metodo statico che verifica se esistono duplicati in una lista di interi.
(Utilizza un metodo per la creazione della lista a partire da un array di interi)

```java
public static boolean verificaDupilicati(List<Integer> lista) {
    boolean trovato=false;
    for(int i=0; i<lista.size(); i++) {
        for(int j= i+1; j<lista.size(); j++) {
            if(lista.get(i)==lista.get(j)) {
                trovato=true;
            }
        }
    }
    return trovato;
}
```

```java
private static List<Integer> listaDi(int...array) {
    List<Integer> lista = new ArrayList<>();
    for(int j =0; j<array.length; j++) {
        lista.add(array[j]);
    }
    return lista;
```

```java
@Test
public void testUnElementoInLista() {
    this.lista.add(2);
    assertFalse(Verifica.verificaDupilicati(listaDi(2)));
}

@Test
public void test2ElementiUgualiInLista() {
    this.lista.add(2);
    this.lista.add(2);
    assertTrue(Verifica.verificaDupilicati(listaDi(2,2)));
}

@Test
public void test2ElementiDiversiInLista() {
    this.lista.add(2);
    this.lista.add(2);
    assertTrue(Verifica.verificaDupilicati(listaDi(2,2,2)));
}
```

1.1) Differenza tra Errore e Fallimento in un test

- **ERRORE**: si presenta quando ho un errore di compilazione nel codice e il test non compila (es. java.lang.Error), oppure quando, a tempo di esecuzione viene sollevata un'eccezione non prevista (checked/unchecked exception).
- **FALLIMENTO:** si presenta quando il codice compila ma il test ha un esito diverso da quello atteso.

1.2) Come evitare di chiamare il costruttore no-args di verifica?

2) Scrivi una classe Studente con matricola e nome e ordina una lista per nome, a parità di nome per matricola (con comparatore esterno)

```java
public List<Studente> ordina(List<Studente> studenti){
    List<Studente> studOrdinati=new ArrayList<Studente>(studenti);
    Collections.sort(studOrdinati,new ComparatoreStudenti());
    return studOrdinati;
}
```

```java
public List<Studente> riempi(Studente ...studenti){
    List<Studente> stud=new ArrayList<Studente>();
    for(Studente s : studenti) {
        stud.add(s);
    }
    return stud;
}

@Test
public void test() {
    Studente s1=new Studente("enea",1);
    Studente s2=new Studente("enea",2);
    List<Studente> students=ordinatore.ordina(riempi(s1,s2));
    assertEquals(s1,students.get(0));
    assertEquals(s2,students.get(1));
}
```

2.2) scrivi il criterio di equivalenza = equals e hashCode.

2.3) Scrivi una SuperClasse Persona con un criterio di ordinamento interno

```java
public class Persona implements Comparable<Persona> {

    private String nome;

    public Persona(String nome, int id) {
        this.nome = nome;
    }

    public String getNome() {
        return this.nome;
    }

    @Override
    public int compareTo(Persona p) {
        return this.getNome().compareTo(p.getNome());
    }
}
```

- e se dovessi implementare un hashset sia di persone che di studenti?

```java
@Override
public boolean equals(Object o) {
    if(o==null || o.getClass()!=this.getClass())
        return false;
    Persona p = (Persona)o;
    return this.getNome().equals(p.getNome());
}
@Override
public int hashCode() {
    return this.getClass().hashCode()+this.getNome().hashCode();
}
```

3) Scrivi test su un metodo a scelta della Java Collections.

3.1) Calcola Collections.max -> è un metodo sovraccarico? In quante varianti esiste?

Collections.max(var) – Collections.max(var, Comparator<>()).

3.2) Scrivi un comparatore anonimo

```java
Collections.max(lista, new Comparator<Studente>() {

    @Override
    public int compare(Studente s1, Studente s2) {
        if( s1.getNome().compareTo(s2.getNome())==0)
            return s1.getMatricola().compareTo(s2.getMatricola());
        return s1.getNome().compareTo(s2.getNome());
    }

});
```

3.3) Differenza tra AssertEquals e AssertSame?

AssertEquals si basa su un uguaglianza decisa dal metodo equals tra due oggetti mutualmente confrontabili

AssertSame controlla l'uguaglianza della cella di memoria

4) Definire una classe di Persona e Studente (come sopra).
Scrivi un metodo che data una lista di studenti List<Studente> ritorna una mappa che associa, per ogni nome, le matricole con quel nome Map<String, List<String>>

```
public class Raggruppatore {

    public static Map<String, List<String>> omonimi(List<Studente> lista){
        Map<String, List<String>> nome2matricole = new HashMap<>();

        for(Studente s : lista) {
            if(nome2matricole.containsKey(s.getNome())) {
                nome2matricole.get(s.getNome()).add(s.getMatricola());
            } else {
                List<String> matricole = new ArrayList<>();
                matricole.add(s.getMatricola());
                nome2matricole.put(s.getNome(), matricole);
            }
        }
        return nome2matricole;
    }
}
```

4.2) Scrivi nelle rispettive classi il metodo toString che stampa:
- per lo studente: classe nome idProgressivo matricola
- per la persona: classe nome idProgressivo

```
public class Persona {

    private int id;
    private static int progId = 0;
    private String nome;

    public Persona(String nome) {
        this.nome = nome;
        this.id = progId++;
    }

    public Persona(String nome, int id) {
        this.nome = nome;
    }

    public int getId() {
        return this.id;
    }

    public String getNome() {
        return this.nome;
    }

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

    @Override
    public String toString() {
        return this.getClass().getSimpleName() + " "+ this.getNome(
                + " "+ this.id ;
    }
}
```

```
public class Studente extends Persona {

    private String matricola;
    private int id;
    private static int progId = 0;

    public Studente(String nome, String matricola) {
        super(nome, 0);
        this.matricola = matricola;
        this.id = progId++;
    }

    @Override
    public int getId() {
        return this.id;
    }

    public String getMatricola() {
        return matricola;
    }

    public void setMatricola(String matricola) {
        this.matricola = matricola;
    }

    @Override
    public String toString() {
        return this.getClass().getSimpleName() + " "+ this.getNome() + " "
                + this.id + " " + this.getMatricola();
    }
}
```

```
public class RaggruppatoreTest {

    private Studente s1;
    private Studente s2;
    private Persona s3;
    private List<Studente> lista;
    private List<Persona> tutti;

    @Before
    public void setUp() throws Exception {
        this.s1 = new Studente("Mario", "A123");
        this.s2 = new Studente("Mario", "B24");
        this.s3 = new Persona("Zorro");
        this.lista = new ArrayList<>();
        this.tutti = new ArrayList<>();
    }

    @Test
    public void testStudenti() {
        lista.add(s1);
        lista.add(s2);
        Map<String, List<String>> mappa =  Raggruppatore.omonimi(lista);
        for(String s : mappa.keySet()) {
            System.out.println("la mappa contiene: " + mappa.get(s));
            System.out.println();
        }
    }
```

```
@Test
public void testTutti() {
    tutti.add(s1);
    tutti.add(s2);
    tutti.add(s3);
    tutti.add(new Persona("Gianni"));
    tutti.add(new Studente("Alessio", "678"));
    for(Persona p : tutti) {
        System.out.println(p.toString());
        System.out.println();
    }
}
```