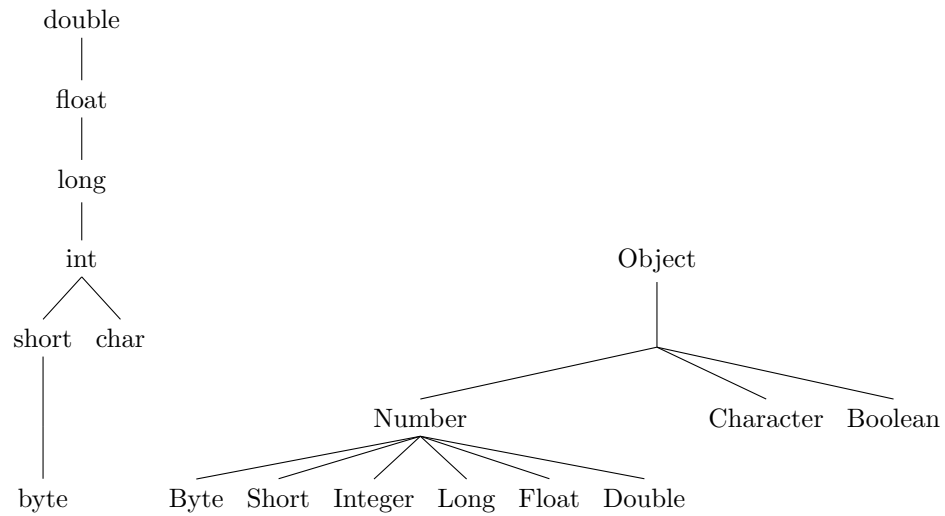


# Java



## 1 Regole

$T_1$  e  $T_2$  sono due tipi:

- $T_1 \leq T_2 \Rightarrow T_1[] \leq T_2[] \leq \text{Object}$
- $T$  primitivo:  $T[] \leq \text{Object}$  e l'unico array compatibile con  $T[]$  è sè stesso

**Esempi:**

- $\text{String}[] \leq \text{Object}[] \leq \text{Object}$
- $\text{Integer}[] \leq \text{Number}[] \leq \text{Object}$
- $\text{Integer}[] \not\leq \text{Long}[]$
- $\text{int}[] \leq \text{Object}$
- $\text{int}[] \not\leq \text{Integer}[]$
- $\text{int}[] \not\leq \text{Object}[]$
- $\text{int}[] \not\leq \text{long}[]$

## 2 Esercizio dei Tipi (tips)

- Se la richiesta ha tipo dinamico ( $P \ p2 = h$ ) allora facciamo "l'intersezione" delle funzioni di  $P$  e  $H$  e diamo la precedenza alle funzioni di  $H$
- Ci potrebbero essere 2 tentativi a runtime quando gli passiamo un oggetto, il primo tentativo è quello che cerca la funzione con firma che possa contenere l'oggetto o dell'oggetto stesso, il secondo tentativo è quello che fa con l'unboxing.

## 3 Tips

```
1 public Costruttore(Tipo... arg) {  
2     // Codice  
3 }
```

Quando ci sono i puntini, possiamo passare un numero variabile di argomenti.

## 4 Esercizi di Visitori

### 4.1 Liste

#### 4.1.1 VisitorTest.java

```
1 package exam2024_01_23.visitor;  
2  
3 public class VisitorTest {  
4     public static void main(String[] args) {  
5         var l1 = new EmptyList(); // la lista []  
6         var l2 = new ListCons(l1, l1); // la lista [][]  
7         var l3 = new ListCons(l1, l2); // la lista [][];  
8         Length length = new Length();  
9         assert l1.accept(length) == 0;  
10        assert l2.accept(length) == 1;  
11        assert l3.accept(length) == 2;  
12    }  
13 }
```

#### 4.1.2 ListCons.java

```
1 package exam2024_01_23.visitor;  
2  
3 import static java.util.Objects.requireNonNull;;  
4  
5 public class ListCons implements ListExp {  
6     private final Exp head; // invariant head != null  
7     private final ListExp tail; // invariant tail != null  
8  
9     public ListCons(Exp head, ListExp tail) {  
10        this.head = requireNonNull(head);  
11    }  
12 }
```

```

11     this.tail = requireNonNull(tail);
12 }
13
14 @Override
15 public <T> T accept(Visitor<T> v) {
16     return v.visitListCons(head, tail);
17 }
18
19 }

```

#### 4.1.3 EmptyList.java

```

1 package exam2024_01_23.visitor;
2
3 public class EmptyList implements ListExp {
4
5     @Override
6     public <T> T accept(Visitor<T> v) {
7         return v.visitEmptyList();
8     }
9
10 }

```

#### 4.1.4 Length.java

```

1 package exam2024_01_23.visitor;
2
3 public class Length implements Visitor<Integer> {
4
5     @Override
6     public Integer visitListCons(Exp head, ListExp tail) {
7         return 1 + tail.accept(this);
8     }
9
10    @Override
11    public Integer visitEmptyList() {
12        return 0;
13    }
14
15 }

```

### 4.2 Classi

#### 4.2.1 VisitorTest.java

```

1 package exam2023_09_20.visitor;
2
3 public class VisitorTest {
4     public static void main(String[] args) {
5         var instance1 = new InstanceEntity();
6         var instance2 = new InstanceEntity();
7         var class1 = new ClassEntity("C1", instance1);
8         var class2 = new ClassEntity("C2", instance2);
9         var class3 = new ClassEntity("C3", class1, class2);
10        assert class3.accept(new SuperClassOf(class3));
11        assert class3.accept(new SuperClassOf(class1));
12        assert class3.accept(new SuperClassOf(class2));

```

```

13     assert !class1.accept(new SuperClassOf(class3));
14     assert !class1.accept(new SuperClassOf(class2));
15     assert !instance1.accept(new SuperClassOf(class3));
16 }
17 }

```

#### 4.2.2 ClassEntity.java

```

1 package exam2023_09_20.visitor;
2
3 import static java.util.Objects.requireNonNull;;
4
5 public class ClassEntity implements JavaEntity {
6     private final String name;
7     private final JavaEntity[] entities; // instances or subclasses
8
9     public ClassEntity(String name, JavaEntity... entities) { //
10         shallow copy
11         this.name = requireNonNull(name);
12         this.entities = requireNonNull(entities);
13     }
14
15     @Override
16     public <T> T accept(Visitor<T> v) {
17         return v.visitClassEntity(name, entities);
18     }
19
20     public String getName() {
21         return name;
22     }
23 }

```

#### 4.2.3 InstanceEntity.java

```

1 package exam2023_09_20.visitor;
2
3 public class InstanceEntity implements JavaEntity {
4
5     @Override
6     public <T> T accept(Visitor<T> v) {
7         return v.visitInstanceEntity();
8     }
9
10 }

```

#### 4.2.4 SuperClassOf.java

```

1 package exam2023_09_20.visitor;
2
3 import static java.util.Objects.requireNonNull;
4
5 public class SuperClassOf implements Visitor<Boolean> {
6     private final ClassEntity classEntity;
7
8     public SuperClassOf(ClassEntity classEntity) {
9         this.classEntity = requireNonNull(classEntity);
10     }

```

```

10 }
11
12 @Override
13 public Boolean visitClassEntity(String name, JavaEntity...
    entities) {
14     if (name.equals(classEntity.getName()))
15         return true;
16     for (var e : entities) {
17         if (e.accept(this))
18             return true;
19     }
20     return false;
21 }
22
23 @Override
24 public Boolean visitInstanceEntity() {
25     return false;
26 }
27
28 }

```

## 5 Esercizi di Iteratori

### 5.1 Liste

#### 5.1.1 Test.java

```

1 package exam2022_06_20.iterators;
2
3 public class Test {
4
5     public static void main(String[] args) {
6         var it = new PowIterator(2, 3);
7         while (it.hasNext())
8             System.out.println(it.next());
9         it.reset(-1, 4);
10        while (it.hasNext())
11            System.out.println(it.next());
12    }
13 }

```

#### 5.1.2 PowIterator.java

```

1 package exam2022_06_20.iterators;
2
3 import java.util.Iterator;
4 import java.util.NoSuchElementException;
5
6 public class PowIterator implements Iterator<Integer> {
7
8     private int base; // invariant: base != 0
9     private int next = 1; // prossimo elemento da restituire
10    private int size; // numero elementi da restituire, nessun
        elemento se size<=0
11
12    protected static int checkBase(int base) {

```

```

13     if (base == 0)
14         throw new IllegalArgumentException("Base cannot be zero");
15     return base;
16 }
17
18 public PowIterator(int base, int size) {
19     this.base = checkBase(base);
20     this.size = size;
21 }
22
23 @Override
24 public boolean hasNext() {
25     return size > 0;
26 }
27
28 @Override
29 public Integer next() {
30     if (!hasNext())
31         throw new NoSuchElementException();
32     var res = next();
33     next *= base;
34     size--;
35     return res;
36 }
37
38 public void reset(int base, int size) {
39     this.base = checkBase(base);
40     this.next = 1;
41     this.size = size;
42 }
43
44 }

```

## 5.2 Classi

### 5.2.1 IteratorTest.java

```

1 package exam2023_07_10.iterator;
2
3 public class IteratorTest {
4     public static void main(String[] args) {
5         var it = new StringArrayRevIterator(new String[] { "a", "b", "c"
6             " });
7         while (it.hasNext())
8             System.out.println(it.next()); // stampa le tre linee c b a
9         it = new StringArrayRevIterator("one", "two", "three");
10        while (it.hasNext())
11            System.out.println(it.next()); // stampa le tre linee three
12            two one
13        it = new StringArrayRevIterator(new String[0]);
14        while (it.hasNext())
15            System.out.println(it.next()); // non stampa nulla
16    }
17 }

```

### 5.2.2 StringArrayRevIterator.java

```

1 package exam2023_07_10.iterator;
2
3 import java.util.Iterator;
4 import java.util.NoSuchElementException;
5
6 import static java.util.Objects.requireNonNull;
7
8 public class StringArrayRevIterator implements Iterator<String> {
9
10     private int index;
11     private final String[] arr;
12
13     public StringArrayRevIterator(String... arr) {
14         this.arr = requireNonNull(arr);
15         index = arr.length - 1;
16     }
17
18     @Override
19     public boolean hasNext() {
20         return index >= 0;
21     }
22
23     @Override
24     public String next() {
25         if (!hasNext())
26             throw new NoSuchElementException();
27         return arr[index--];
28     }
29
30 }

```