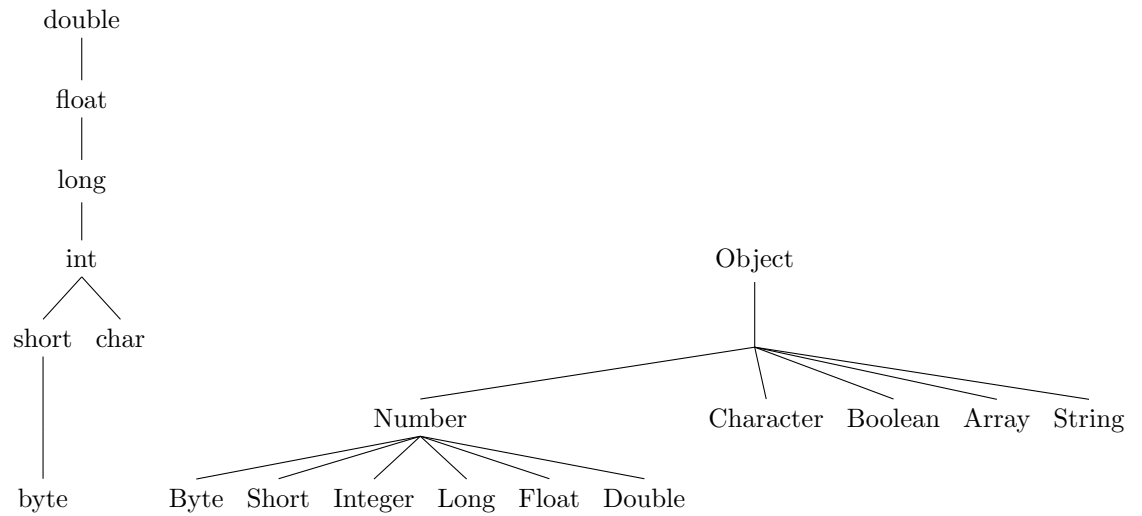


# Java



## 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 Definizioni

- **Boxing:** è il processo di conversione di un tipo primitivo nella classe oggetto wrapper corrispondente (es. `int` in `Integer`)
- **Unboxing:** è il processo contrario del boxing, ovvero la conversione di un oggetto wrapper in un tipo primitivo (es. `Integer` in `int`)

## 3 Esercizio dei Tipi (tips)

- Se la richiesta ha tipo dinamico (`P p2 = h`) allora guardiamo le funzioni della classe 'statica' (`P`) e le funzioni che ha in comune con la classe dinamica (`H`) verranno sovrascritte dalla classe dinamica.
- 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.
- Se la richiesta ha tipo `H` che estende `P` allora guardiamo le funzioni di `H` e quelle di `P` dove quelle di `H` fanno override di quelle di `P`.

## 4 Tips

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

Quando ci sono i puntini, possiamo passare un numero variabile di argomenti. `arg` è considerato come array.

## 5 Esercizi di Visitori Liste

### 5.0.1 VisitorTest.java

```
package exam2024_01_23.visitor;  
public class VisitorTest {  
    public static void main(String[] args) {  
        var l1 = new EmptyList(); // la lista []  
        var l2 = new ListCons(l1, l1); // la lista [][]  
        var l3 = new ListCons(l1, l2); // la lista [][];  
        Length length = new Length();  
        assert l1.accept(length) == 0;  
        assert l2.accept(length) == 1;  
        assert l3.accept(length) == 2;  
    }  
}
```

### 5.0.2 ListCons.java

```
package exam2024_01_23.visitor;
import static java.util.Objects.requireNonNull;
public class ListCons implements ListExp {
    private final Exp head; // invariant head != null
    private final ListExp tail; // invariant tail != null
    public ListCons(Exp head, ListExp tail) {
        this.head = requireNonNull(head);
        this.tail = requireNonNull(tail);
    }
    @Override
    public <T> T accept(Visitor<T> v) {
        return v.visitListCons(head, tail);
    }
}
```

### 5.0.3 EmptyList.java

```
package exam2024_01_23.visitor;
public class EmptyList implements ListExp {
    @Override
    public <T> T accept(Visitor<T> v) {
        return v.visitEmptyList();
    }
}
```

### 5.0.4 Length.java

```
package exam2024_01_23.visitor;
public class Length implements Visitor<Integer> {
    @Override
    public Integer visitListCons(Exp head, ListExp tail) {
        return 1 + tail.accept(this);
    }
    @Override
    public Integer visitEmptyList() {
        return 0;
    }
}
```

## 6 Esercizi di Visitatori Classi

### 6.0.1 VisitorTest.java

```
package exam2023_09_20.visitor;

public class VisitorTest {
    public static void main(String[] args) {
        var instance1 = new InstanceEntity();
        var instance2 = new InstanceEntity();
        var class1 = new ClassEntity("C1", instance1);
        var class2 = new ClassEntity("C2", instance2);
        var class3 = new ClassEntity("C3", class1, class2);
        assert class3.accept(new SuperClassOf(class3));
        assert class3.accept(new SuperClassOf(class1));
    }
}
```

```

        assert class3.accept(new SuperClassOf(class2));
        assert !class1.accept(new SuperClassOf(class3));
        assert !class1.accept(new SuperClassOf(class2));
        assert !instance1.accept(new SuperClassOf(class3));
    }
}

```

### 6.0.2 ClassEntity.java

```

package exam2023_09_20.visitor;
import static java.util.Objects.requireNonNull;
public class ClassEntity implements JavaEntity {
    private final String name;
    private final JavaEntity[] entities; // instances or subclasses
    public ClassEntity(String name, JavaEntity... entities) { //
        shallow copy
        this.name = requireNonNull(name);
        this.entities = requireNonNull(entities);
    }
    @Override
    public <T> T accept(Visitor<T> v) {
        return v.visitClassEntity(name, entities);
    }
    public String getName() {
        return name;
    }
}

```

### 6.0.3 InstanceEntity.java

```

package exam2023_09_20.visitor;
public class InstanceEntity implements JavaEntity {
    @Override
    public <T> T accept(Visitor<T> v) {
        return v.visitInstanceEntity();
    }
}

```

### 6.0.4 SuperClassOf.java

```

package exam2023_09_20.visitor;
import static java.util.Objects.requireNonNull;
public class SuperClassOf implements Visitor<Boolean> {
    private final ClassEntity classEntity;
    public SuperClassOf(ClassEntity classEntity) {
        this.classEntity = requireNonNull(classEntity);
    }
    @Override
    public Boolean visitClassEntity(String name, JavaEntity...
        entities) {
        if (name.equals(classEntity.getName()))
            return true;
        for (var e : entities) {
            if (e.accept(this))
                return true;
        }
        return false;
    }
}

```

```

    }
    @Override
    public Boolean visitInstanceEntity() {
        return false;
    }
}

```

## 7 Esercizi di Iteratori

### 7.1 Liste

#### 7.1.1 Test.java

```

package exam2022_06_20.iterators;

public class Test {

    public static void main(String[] args) {
        var it = new PowIterator(2, 3);
        while (it.hasNext())
            System.out.println(it.next());
        it.reset(-1, 4);
        while (it.hasNext())
            System.out.println(it.next());
    }
}

```

#### 7.1.2 PowIterator.java

```

package exam2022_06_20.iterators;

import java.util.Iterator;
import java.util.NoSuchElementException;

public class PowIterator implements Iterator<Integer> {

    private int base; // invariant: base != 0
    private int next = 1; // prossimo elemento da restituire
    private int size; // numero elementi da restituire, nessun
        elemento se size<=0

    protected static int checkBase(int base) {
        if (base == 0)
            throw new IllegalArgumentException("Base cannot be zero");
        return base;
    }

    public PowIterator(int base, int size) {
        this.base = checkBase(base);
        this.size = size;
    }

    @Override
    public boolean hasNext() {
        return size > 0;
    }
}

```

```

@Override
public Integer next() {
    if (!hasNext())
        throw new NoSuchElementException();
    var res = next;
    next *= base;
    size--;
    return res;
}

public void reset(int base, int size) {
    this.base = checkBase(base);
    this.next = 1;
    this.size = size;
}
}

```

## 7.2 Classi

### 7.2.1 IteratorTest.java

```

package exam2023_07_10.iterator;

public class IteratorTest {
    public static void main(String[] args) {
        var it = new StringArrayRevIterator(new String[] { "a", "b", "c"
        " });
        while (it.hasNext())
            System.out.println(it.next()); // stampa le tre linee c b a
        it = new StringArrayRevIterator("one", "two", "three");
        while (it.hasNext())
            System.out.println(it.next()); // stampa le tre linee three
        two one
        it = new StringArrayRevIterator(new String[0]);
        while (it.hasNext())
            System.out.println(it.next()); // non stampa nulla
    }
}

```

### 7.2.2 StringArrayRevIterator.java

```

package exam2023_07_10.iterator;

import java.util.Iterator;
import java.util.NoSuchElementException;

import static java.util.Objects.requireNonNull;

public class StringArrayRevIterator implements Iterator<String> {

    private int index;
    private final String[] arr;

    public StringArrayRevIterator(String... arr) {
        this.arr = requireNonNull(arr);
    }
}

```

```
        index = arr.length - 1;
    }

    @Override
    public boolean hasNext() {
        return index >= 0;
    }

    @Override
    public String next() {
        if (!hasNext())
            throw new NoSuchElementException();
        return arr[index--];
    }
}
```