Polimorfismo parametrico
vs
polimorfismo per inclusione

Esercizio

- Definire il tipo di dato “Stack” con operazioni
  - Push(element)
  - Pop()
  - Non “forzare” una specifica implementazione
  - Non “forzare” un tipo specifico per gli elementi
    - È una libreria: non sapete come la useranno
- Definire l'implementazione a lista concatenata:
  - LinkedStack
- Usate al meglio i costrutti di Java!

Una soluzione

- Prima parte

```
public interface ObjectStack {
   public void push( Object el );
   public Object pop() throws EmptyStackException;
}
```

```
public class EmptyStackException extends Exception {
       static final long serialVersionUID = 99999;
}
```

Una soluzione

- Seconda parte

```
public class LinkedObjectStack implements ObjectStack {
        private class StackRecord {
                Object el;
                StackRecord next;
                StackRecord( Object el, StackRecord next ){  this.el = el;   this.next = next;  }
        }
        StackRecord top;

        public void push( Object el ) {  top = new StackRecord( el, top );  }

        public Object pop() throws EmptyStackException {
                Object res;
                if( top == null ) throw new EmptyStackException();
                res = top.el;
                top = top.next;
                return res;        }
}
```

Una soluzione

- Esempio di uso #1

```
ObjectStack so = new LinkedObjectStack();

so.push(1);     // conversione implicita a Integer (autoboxing)
so.push(2);     // e upcast automatico a Object
so.push(3);

try{
    while( true ){
        System.out.println( ((Integer)so.pop()).intValue() ); }
}
catch( Exception e ) { System.out.println( “fine” ); }
```

Se non faccio il downcast...

non posso usare questo metodo (specifico di Integer)

```
3
2
1
fine
```

output

Una soluzione

- Esempio di uso #2

```
ObjectStack so = new LinkedObjectStack();

so.push(1);     // conversione implicita a Integer (autoboxing)
so.push(2);     // e upcast automatico a Object
so.push("A");

try{
    while( true ){
        System.out.println( ((Integer)so.pop()).intValue() ); }
}
catch( Exception e ) { System.out.println( "fine" ); }
```

Se mi distraggo compila ancora ma...

output

```
java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer
```

Una soluzione migliore: template

- Prima parte

```
public interface GenericStack<ElemType> {
	public void push( ElemType el );
	public ElemType pop() throws EmptyStackException;
}
```

Una soluzione migliore: template

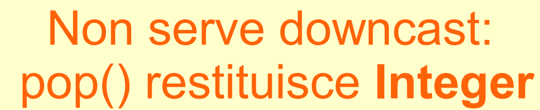
- Seconda parte

```
public class GenericLinkedStack<ElemType> implements GenericStack<ElemType> {
        private class StackRecord<ElType> {
           ElType el;
           StackRecord<ElType> next;
           StackRecord( ElType el, StackRecord<ElType> next ){ this.el = el; this.next = next; }
        }
        StackRecord<ElemType> top;

        public void push( ElemType el ) {  top = new StackRecord<ElemType>( el, top ); }
        public ElemType pop() throws EmptyStackException {
                ElemType res;
                if( top == null ) throw new EmptyStackException();
                res = top.el;
                top = top.next;
                return res;
        }
}
```

Una soluzione migliore: template

- Esempio di uso #1

```
GenericStack< Integer > si = new GenericLinkedStack< Integer >();

si.push(1);
si.push(2);
si.push(3);
try{
	while( true ){ System.out.println( si.pop().intValue() ); }
}
catch( Exception e ) { System.out.println( “fine” ); }
```




```
3
2
1
fine
```

output

Una soluzione migliore: template

- Esempio di uso #2

```
GenericStack< Integer > si = new GenericLinkedStack< Integer >();

si.push(1);
si.push(2);
si.push(“A”);
try{
    while( true ){ System.out.println( si.pop().intValue() ); }
}
catch( Exception e ) { System.out.println( “fine” ); }
```

push(**Integer**)

```
1. ERROR in mylists/Test2.java (at line 8)
        si.push("A");
           ^^^^
The method push(Integer) in the type GenericStack<Integer> is not applicable
for the arguments (String)
```

output **del compilatore!**

Confronto

- Pro del polimorfismo parametrico:
  - Non richiede controlli di tipo a run time
  - Anticipa scoperta errori di tipo a tempo di compilazione
- Pro del polimorfismo per inclusione
  - Permette strutture dati eterogenee
  - Ad esempio, Stack di elementi di tipo diverso

Prendere il meglio

- In molti casi
  - Servono collezioni di elementi eterogenei
  - Ma vogliamo usarli allo stesso modo
- Esempio:
  - Una scena è una *lista* di forme eterogenee (rettangoli, ellissi, linee, ecc.)
  - Vogliamo usarla per implementare *refresh*, che deve solo inviare un messaggio *draw()* a tutte le forme della lista
- Quindi
  - Identificare le modalità d'uso degli elementi
  - Scegliere una superclasse comune o fattorizzarle in una interfaccia
  - Usare la superclasse/interfaccia come argomento del template

Note sull'implementazione

- In Java – internamente - sarebbe comunque uno stack di Object
  - Ma coi template dichiariamo che vogliamo metterci solo oggetti di un certo tipo
  - Per questo non possiamo legare il parametro a tipi elementari come int o float ma dobbiamo usare i wrapper
- In C++ ogni uso dei template fa compilare una nuova classe
  - Sorta di macro
  - Il compilatore inserisce nel programma la definizione di classe specializzata e la ricompila per ogni parametro attuale