Cenni su programmazione con tipi generici (generics)

Tipi generici (generics)

- Programmazione generica:

  Creazione di costrutti che possono essere utilizzati con tipi di dati diversi
  - Es. ArrayList<String>, ArrayList<Rectangle>,etc.
- Permette riutilizzo del codice
- Si può realizzare con:
  - ereditarietà (variabili di tipo Object)
  - variabili di tipo (variabili a cui si può assegnare un tipo non primitivo e che possono essere usate come tipi nelle dichiarazioni)

Realizzazione di classi con tipi generici

- I tipi generici
  - sono dichiarati tra parentesi angolari “<“ e “>” dopo il nome della classe
  - di solito sono indicati con una lettera maiuscola
  - sono utilizzati come tipi per dichiarare le variabili, i parametri dei metodi e il valore di restituzione di un metodo nel codice della classe
- Es.: una classe generica Pair che contiene coppie di oggetti

Realizzazione di classi con parametri

```
public class Pair<S,T>{
    public Pair(S primoEl, T secondoEl) {
      primo = primoEl;
      secondo = secondoEl;
    }

  public S getFirst() { return primo; }
  public T getSecond() { return secondo; }
  private S primo;
  private T secondo;
}
```

- Pair<S,T> definisce un tipo generico
- Una classe può usare più variabili di tipo nella sua definizione, ma ogni tipo deve essere unico
  - Pair<T,T> genera errore sulla seconda T

Classe tester per Pair

```
public class PairTester {
    public static void main(String[] args) {
      Pair<Double,Integer> p =
                    new Pair<Double,Integer>(3.0,3);

      double x = p.getFirst();
      int y = p.getSecond();
    }
}
```

- L'effetto ottenuto è come se le variabili di tipo venissero assegnate con i tipi indicati al momento dell'istanziazione
  - in questo caso, S con Double e T con Integer

Metodi generici

- Possono appartenere anche a classi non generiche
- Considera l'esempio:

```
public class ArrayUtil{
  public static String print(String[] a){
        String s="";
        for(String e : a)
              s+=e+" ";
        s+='\n';
        return s;
   }
   ……
}
```

- Questo algoritmo può essere riutilizzato per convertire in String un array di oggetti di qualsiasi tipo

Metodi generici

- Stampa array di oggetti di tipo arbitrario:

```
public class ArrayUtil{
  public static <E> String print(E[] a){
        String s="";
        for(E e : a)
              s+=e+" ";
        s+='\n';
        return s;
   }
   ……
}
```

- Utilizzo metodo:

```
Rectangle[ ] rectangles= ……;
ArrayUtil.print(rectangles);
```

Osservazioni

- Per utilizzare metodo generico, non occorre specificare il tipo effettivo da assegnare alle variabili di tipo
- Il tipo del parametro è dedotto dal compilatore dall'uso che ne facciamo
  - Nell'esempio, il compilatore deduce che Rectangle è il tipo effettivo da usare per E

Limiti al tipo delle variabili di tipo

- Può essere necessario limitare variabili di tipo
- Es: un metodo generico min va bene per oggetti che possono essere confrontati

```
public static <E> E min(E[] a) .........
```

- Non pone alcun vincolo sul tipo che possiamo assegnare ad E

Uso di extends per tipi generici

```
public static <E extends Comparable> E min(E[] a) {
   E smallest = a[0];
   for (int i=0; i<a.length;i++)
       if (a[i].compareTo(smallest)<0)
              smallest = a[i];
   return smallest;
}
```

- In questo caso E può essere assegnata con un qualsiasi tipo che è compatibile con Comparable (sotto-tipo di Comparable)
- Per esprimere più di un vincolo si usa “&”

```
public static <E extends Comparable & Cloneable> E
   min(E[] a) .........
```

Approfondimento sui generics

- Le variabili di tipo non sono tipi di Java
  - Ad esempio, se T è una variabile di tipo come in Pair<S,T>, non troveremo mai T.java oppure T.class nel file system
  - T non fa parte del nome della classe Pair
- Nella compilazione di Pair<S,T> si genera il file Pair.class dove non vi è traccia dei parametri T e S (type erasure)
  - La classe viene trasformata nella classe "grezza" corrispondente

Type erasure

- Consente alle applicazioni Java che usano tipi generici di essere compatibili con le librerie e le applicazioni create prima dell'avvento dei tipi generici (Java 5.0) .
- Tutta l'informazione relativa ai tipi generici viene rimossa
- Ciascuna variabile di tipo viene sostituita con un tipo effettivo di Java opportuno (Object oppure il tipo che delimita lo scope della variabile attraverso extends)
- I tipi generici nel codice vengono rimpiazzati con il tipo grezzo corrispondente (ad es. ArrayList<BankAccount> è rimpiazzato con ArrayList)
- Vengono aggiunti i casting dove è necessario (deducendoli dalle istanziazioni degli oggetti)
  - prima della compilazione in senso stretto avviene una traduzione code-to-code

Classe Pair<S,T> dopo type erasure

```
public class Pair{
    public Pair(Object primoEl, Object secondoEl) {
      primo = primoEl;
      secondo = secondoEl;
    }

  public Object getFirst() { return primo; }
  public Object getSecond() { return secondo; }
  private Object primo;
  private Object secondo;
}
```

- Pair è il tipo grezzo corrispondente a Pair<S,T>

Classe PairTester dopo type erasure

```
public class PairTester {
    public static void main(String[] args) {
      Pair p = new Pair(3.0,3);

      double x = (Double) p.getFirst();
      int y = (Integer) p.getSecond();
    }
}
```

- Cancellazione dei tipi generici e aggiunta di operatori casting in maniera opportuna

Metodo min dopo type erasure

```
public static Comparable min(Comparable[] a) {
   Comparable smallest = a[0];
   for (int i=0; i<a.length;i++)
       if (a[i].compareTo(smallest)<0)
              smallest = a[i];
   return smallest;
}
```

- Comparable è il tipo meno generale che è compatibile con tutti i tipi che possono essere utilizzati con il metodo

Type erasure (bound multipli)

```
public static <E extends Comparable & Iterable>
        E min(E[] a) {
                E smallest = a[0]; smallest.iterator();
                for (int i=0; i<a.length;i++)
                        if (a[i].compareTo(smallest)<0)
                                smallest = a[i];
                return smallest;
        }
```

- dopo type erasure:

```
public static Comparable min(Comparable a[]) {
    Comparable smallest = a[0];      ((Iterable)smallest).iterator();
    for (int i = 0; i < a.length; i++)
      if (a[i].compareTo(smallest) < 0)  smallest = a[i];
    return smallest;
  }
```

Esempio: verifica type erasure

```
ArrayList<Integer> li = new ArrayList<Integer>();
ArrayList<Float> lf = new ArrayList<Float>();
if (li.getClass() == lf.getClass()){ // evaluates to true
    System.out.println("Equal");
    System.out.println(li.getClass().getName());
}
```

- Stampa nella console il messaggio:

  Equal

  java.util.ArrayList

Decompilazione bytecode

- Esistono dei tool per risalire dal bytecode al codice sorgente
  - Ad esempio JAD

- Si possono verificare gli effetti della type erasure seguendo questi passi:
  - Scrivere un sorgente con generics
  - Generare il bytecode
  - Decompilare

Osservazioni finali

- Per effetto della type erasure non vi è modo di risalire a runtime al tipo dell'oggetto che effettivamente viene usato in una classe generica

- Ad es., a runtime in Point<S,T> viene perso ogni riferimento a S e T e quindi non è possibile usare S e T per operazioni a runtime tipo:
  - istanziazioni di oggetti
  - casting
  - controlli di tipo

Errori con uso variabili di tipo

```
public class MyClass<E> {
    public static void myMethod(E item) {
        if (item instanceof E) {
            //Compiler error ...
        }
        E item2 = new E(); //Compiler error
        E[ ] iArray = new E[10]; //Compiler error
        E obj = (E) new Object();
            //Unchecked cast warning
    }
}
```