Java Generics

SOftEng
http://softeng.polito.it

Problems

- Often one needs the same behavior for different kind of classes
  - A typical solution is to use Object references to accommodate any object type
  - The use of Object references brings cumbersome code
- Alternative?
  - Use Java Generics (“generic” type) for methods, attributes and classes

Example

- One may need to represent ID of persons in different forms

```
public class Person {
  String first; String last; Object ID;

 Person(String f, String l, Object ID){
     this.first = f;
     this.last = l;
     this.ID = ID;
   }
  }
```

Example

- It is possible can use it with different types

```
Person a = new Person("Al","A",new
                                Integer(123));
Person b = new Person("Pat","B","s32");
```

- Casts may be needed …

```
Integer id = (Integer) a.getID();
```

- … that may be dangerous

```
Integer id = (Integer) b.getID();
```

ClassCastException at run-time

Generic class

```
public class Person<T> {
   String first;
     String last;
     T ID;
   Person(String first,String last,T ID){
       this.first = first;
       this.last = last;
       this.ID = ID;
     }
   T getID(){ return ID; }
  }
```

Generics use

- Declaration is longer but ...

```
Person<Integer> a = new Person<Integer>
                ("Al","A",new Integer(123));
Person<String> b = new Person<String>
                              ("Pat","B","s32");
```

- ... use is more compact and safer

```
Integer id1 = a.getID();
   String id2 = b.getID();
   Integer ids = b.getID();
```

Compiler error:
type mismatch

Generic type declaration

- Syntax:

```
(class|interface) Name <P1 {,P2}>
```

- Type parameters, e.g., `P1`
  - Conventionally uppercase letter
  - Represents classes or interfaces
  - Usually: T(ype), E(lement), K(ey), V(alue)

Generic collections

- All collection interfaces and classes have been redefined using Generics
- Use of Generics lead to code that is
  - safer
  - more compact
  - easier to understand
  - equally performing

Generic list – excerpt

```
public interface List<E>{
   void add(E x);
   Iterator<E> iterator();
}
public interface Iterator<E>{
   E next();
   boolean hasNext();
}
```

Example

- Using a list of Integers
  - Without generics (`ArrayList list`)

```
list.add(0, new Integer(42));
int n= ((Integer)(list.get(0))).intValue();
```

  - With generics (`ArrayList<Integer> list`)

```
list.add(0, new Integer(42));
int n= ((Integer)(list.get(0))).intValue();
```

  - + autoboxing (`ArrayList<Integer> list`)

```
list.add(0,new Integer(42));
int total = list.get(0).intValue();
```

Diamond operator

- Reference type parameter must match the class parameter used in instantiation

```
List<String> l=new
                LinkedList<String>();
```

Generic method declaration

- Syntax:

  modifiers **<T> ret_*type* name(pars)**

- **pars** can be:
  - as usual
  - **T**
  - **type<T>**

Generics classes

- There is only one class generated (by the compiler) for each generic type declaration

```
Person<Integer> a = new Person<Integer>
                ("Al","A",new Integer(123));
Person<String> b = new Person<String>
                              ("Pat","B","s32");
boolean same=(a.getClass()==b.getClass());
```

believe it or not
same is *true*