Generics

Hsuan-Tien Lin

Deptartment of CSIE, NTU

OOP Class, May 19, 2009

How can we write a class for an Integer set of arbitrary size?

```
class IntegerSet{
  Integer[] arr;
  int count;
  IntegerSet(int len){  arr = new Integer[len]; }
  void add(Integer i){
    if (count < arr.length){arr[count] = i; count++; }
  }
  void removeLast(){ arr[count] = null; count--; }
}
```

How can we write a class a String set of arbitrary size?

```
class StringSet{
  String[] arr;
  int count;
  StringSet(int len){  arr = new String[len]; }
  void add(String i){
    if (count < arr.length){arr[count] = i; count++; }
  }
  void removeLast(){ arr[count] = null; count--; }
}
```

① 寫 ANYSet.java

② 取代 ANY 成為 ...

How can we write classes for Integer/String/Double/Professor sets of arbitrary size?

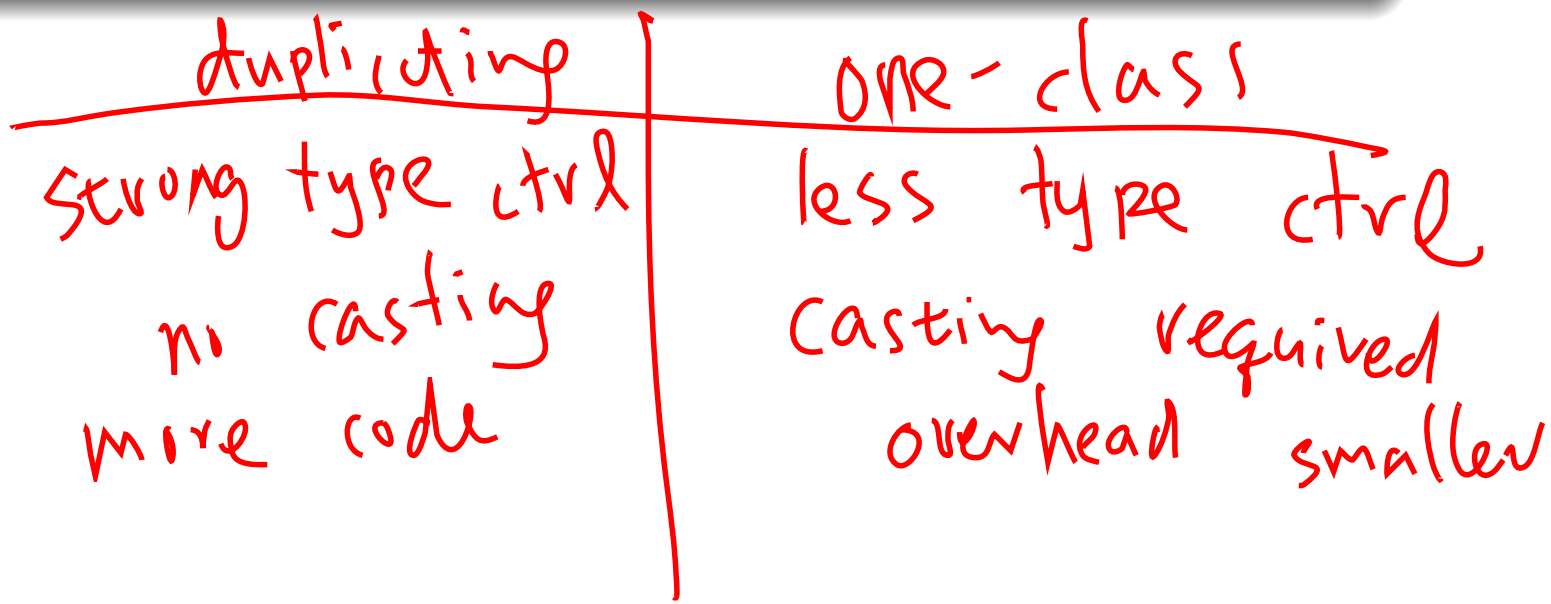
```
class ObjectSet{
  Object[] arr; int count;
  ObjectSet(int len){  arr = new Object[len]; }
  void add(Object o){   ... }
}
```

template (容器技巧)

How can we write **one class** for arbitrary sets of arbitrary size?

```
class PowerfulSet{
  String[] sarr; Integer[] iarr; Professor[] parr;
  int scount, icount, pcount;
  PowerfulSet(int len){  sarr = new String[len]; iarr ...;
parr....}
  void add(String s){   ... }
  void add(Integer i){ ...}
}
```

How does duplicating solution compare with one-class solution?



```
class StringSet extends ObjectSet{
  StringSet(int len){ super(len); }
  void add(String s){ super.add(s); }
  String get(int i){ return (String)super.get(i); }
}
```

How can we write one class for arbitrary sets of arbitrary size **while keeping type information**?

```
class Set<T>{
  Object[] arr;
  int count;
  Set(int len){  arr = new Object[len]; }
  void add(T i){if (count < arr.length)
{arr[count] = i; count++; }}
  T get(int pos){ return (T)arr[pos]; }
}
```

是

不是

行为(行为物)

string is Object

不

正

Should StringSet extend ObjectSet?

```
class StringSet extends ObjectSet{
  StringSet(int len){ super(len); }
  void add(String s){ super.add(s); }
  String get(int i){ return (String)super.get(i); }
}
```

Java Solution: Generics (since 1.4)

- no manual duplicating (as opposed to old languages): save coding efforts
- no automatic duplicating (as opposed to C++): save code size and re-compiling efforts
- check type information very strictly by compiler (as opposed to single-object polymorphism): ensure type safety in JVM

Note: type information **erased** after compilation

AbstractCollection.containsAll(...)

Go Check the Source Code!

AbstractCollection.addAll(...)

Go Check the Source Code!

AbstractCollection.toArray(...)

Go Check the Source Code!

More on Type Erasure

3. false true

```
1  ArrayList<String> l1 = new ArrayList<String>();
2  ArrayList<Integer> l2 = new ArrayList<Integer>();
3  System.out.println(l1.getClass() == l2.getClass());
4  System.out.println(l1 instanceof Collection<String>);
```

4. false true

More on Type Safety

```
1  ArrayList<String>[] ls1 = new ArrayList<String>[10];
2  ArrayList<?>[]      ls2 = new ArrayList<?>[10];
3  ArrayList<String>[] ls3 = new ArrayList<?>[10];
```

Why Is This Illegal?

```
1  <T> T[] makeArray(T t, int len){
2    return new T[len];
3  }
```