Raw type

In 2004, Java 5 came out, with *generics* added to it. Class ArrayList had been *genericized*, so it was declared like this:

public class ArrayList<E> extends AbstractList<E> implements … { … }

From then on, one could use *generic types* like AbstractList<String> and AbstractList<JFrame>. Programming became more "type safe" and easier.

But for backward-compatibility, all the code written using Java 4 had to work in Java 5. For example, the new expression new ArrayList() had to still be syntactically correct and work as it did before.

Definition: The *raw type* is the generic type without any arguments.

For example, ArrayList<String> and ArrayList<JFrame> are generic types, while ArrayList is a raw type. Whenever it is possible, we urge you to follow this:

Strong suggestion: Don't use the raw type of a generic class. Example: Don't use ArrayList by itself unless you really have to.

Below, we discuss what may happen when using a raw type.

Using a raw type

You can mix uses of raw types and generic types, but the compiler will give a warning if it cannot tell whether a statement or expression is type safe.

Consider the code to the right, where b's type is a raw type. A warning is given on the call to b.add because it cannot be determined what type of elements are allowed in b. The warning is: "unchecked call to add(E) as a member of the raw type java.util.ArrayList". There is no warning for the fifth line, but it's obvious that a class cast exception will be thrown at runtime because b.get(0) is not an Integer.

```
ArrayList b;
b= new ArrayList<String>();
b= new ArrayList();
b.add("abc");  // unchecked warning
Integer s= (Integer) b.get(0);
```

With the code to the right, the compiler issues as unchecked conversion warning because a raw-type c is stored in variable b of type ArrayList<String>, and it is not known what values are in ArrayList c. It's the programmer's duty to known that c's array elements are only of type <String> (or null).

```
ArrayList c= new ArrayList();
ArrayList<String> b= c; // unchecked
                        // conversion
b.add("abc")
```

This last example was culled from a much larger program and changed to illustrate two points. In class W<E>, field f is public, so it can be changed either by storing directly into it or by calling setter method set.

The type of parameter of function M.test is raw type W. Warnings for the two statements in the body of method test are given because it cannot be determined that the type of value assigned to f is E:

Warning: unchecked call to set(E) as a member of the raw type W
Warning: unchecked assignment to variable f as member of raw type W

Change the declaration of parameter w of method test to the following and the warnings disappear.

W<M> w

```
class W<E> {
   public E f;
   public void set(E p) {f= p;}
}

class M {
   public void test(W w) {
      w.set(new M());  //warning
      w.f= new M();    // warning
    }
}
```