

### Generic count # of elements in a collection

```
public final class
Algorithm {
    public static
    <T> int countIf(Collection<T> c, UnaryPredicate<T> p) {
        int count
        = 0;
        for (T
        elem : c)
            if (p.test(elem))
                ++count;
        return
        count;
    }
}
public interface
UnaryPredicate<T> {
    public boolean
    test(T obj);
}
import java.util.*;
class OddPredicate
implements UnaryPredicate<Integer> {
    public boolean
    test(Integer i) {
        return i % 2 != 0;
    }
}
public class Test {
    public static
    void main(String[]
    args) {
        Collection<Integer> c
        = Arrays.asList(1, 2,
        3, 4);
```

### Generic count # of elements in a collection (cont)

```
> int count = Algorithm.countIf(c, new OddPredicate());
    System.out.println("Number of odd integers = " + count);
}
```

The program prints:  
Number of odd integers = 2

### compile? If not, why?

```
public class
Singleton<T> {
    public static T
    getInstance() {
        if
        (instance == null)
            instance = new Singleton<T>();
        return
        instance;
    }
    private static T
    instance = null;
}
```

No. You cannot create a static field of the type parameter T.

### Swap positions of two elements in array.

```
public final class
Algorithm {
```

### Swap positions of two elements in array. (cont)

```
> public static<T> void
swap(T[] a, int i, int j) {
    T temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
}
```

### Method 2 find the maximal element of a list.

```
import java.util.*;
public final class
Algorithm {
    public static<T>
    extends Object &
    Comparable<? super T>>
    T
    max(List<? extends T>
    list, int begin, int
    end) {
        T maxElem
        = list.get(begin);
        for
        (++begin; begin < end;
        ++begin)
            if (maxElem.compareTo(list.get(begin)) < 0)
                maxElem =
                list.get(begin);
        return
        maxElem;
    }
}
```

### How invoke 2 find the first integer...

```
public static<T>
int findFirst(List<T> list, int
begin, int end, UnaryPredicate<T> p)
import java.util.*;
public final class
Algorithm {
    public static
    <T>
    int
    findFirst(List<T>
    list, int begin, int
    end, UnaryPredicate<T> p) {
        for (
        begin < end; ++begin)
            if (p.test(list.get(begin)))
                return begin;
        return -
        1;
    }
    // x > 0 and y >
    0
    public static
    int gcd(int x, int y) {
        for (int
        r; (r = x % y) != 0; x =
        y, y = r) {}
        return y;
    }
}
```



By jimisanchez

Published 2nd May, 2016.

Last updated 8th May, 2016.

Page 1 of 2.

Sponsored by [Readable.com](http://readable.com)

Measure your website readability!

<https://readable.com>

### Compiler erases parameters, y use generics?

The Java compiler enforces tighter type checks on generic code at compile time.

Generics support programming types as parameters.

Generics enable you to implement generic algorithms.

### Converted to after type erasure?

```
public class Pair {
    public Pair(Object key, Object value) {
        this.key = key;
        this.value = value;
    }
    public Object getKey() { return key; }
    public Object getValue() { return value; }
    public void setKey(Object key) { this.key = key; }
    public void setValue(Object value) { this.value = value; }
    private Object key;
    private Object value;
}
```

### converted to after type erasure?

```
public static <T extends Comparable<T>>
    int findFirstGreaterThan(T[] at, T elem) {
        // ...
    }
// becomes
public static int findFirstGreaterThan(Comparable[] at, Comparable elem) {
    // ...
}
```

### compile? If not, why?

```
public static void print(List<? extends Number> list) {
    for (Number n : list)
        System.out.println(n + " ");
    System.out.println();
}
```

Yes

### Will the following class compile? If not, why?

```
public final class Algorithm {
    public static <T> T max(T x, T y) {
        return x > y ? x : y;
    }
}
```

No. The greater than (>) operator applies only to primitive numeric types.

### Compile?

```
class Node<T> implements Comparable<T> {
    public int compareTo(T obj) { / ... / }
}
```

Yes.

### Compile

```
class Shape { / ... / }
class Circle extends Shape { / ... / }
class Rectangle extends Shape { / ... / }
class Node<T> { / ... / }
Node<Circle> nc = new Node<>();
Node<Shape> ns = nc;
```

No. Because Node<Circle> is not a subtype of Node<Shape>.

