

Example - Monomorphic Calls

```
public class Main implements Observer {  
    public static void main(String[] args) {  
        Main m = new Main(); → <init>  
        Subject s = new Subject(); → <init>  
        s.addObserver(m); → add Observer  
        s.modify();  
    }  
    public void update(Observable o, Object arg) {  
        System.out.println(o+" notified me!");  
    }  
    static class Subject extends Observable {  
        public void modify() {  
            setChanged();  
            notifyObservers();  
        }  
    }  
}
```

Example - Polymorphic Calls

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Collection c = makeCollection(args[0]);
        c.add(args[1]);
    }
    static Collection makeCollection(String s) {
        if(s.equals("list")) {
            return new ArrayList();
        } else {
            return new HashSet();
        }
    }
}
```

The diagram illustrates a polymorphic call. A blue arrow points from the `makeCollection` method call in the `main` method to the `makeCollection` method definition. The `makeCollection` method is annotated with blue circles around the `static` keyword, the `Collection` return type, the `if(s.equals("list"))` condition, and the `return new` statements, highlighting its polymorphic nature.

Rapid Type Analysis - example

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Collection c = makeCollection(args[0]);
        c.add("x");
        new LinkedList();
    }
    static Collection makeCollection(String s) {
        if(s.equals("list")) {
            return new ArrayList();
        } else {
            return new HashSet();
        }
    }
}
```

XTA - example

```
class Main {  
    static Collection c;  
    static Object o; { AL, String,  
    public static void main(String[] args) {  
        c = initC1();  
        set0(c);  
        c = initC2(new String("x"));  
        process();  
    }  
    static Collection initC1() {  
        return new ArrayList<Object>(o);  
    }  
    static Collection initC2(Object o) { ← { AL  
        List<Object> l = new ArrayList<Object>(o);  
        return l;  
    }  
    static String set0(Object o){  
        Main.o = o;  
        o.toString();  
    }  
    static void process() {  
        List<Object> l = new LinkedList<Object>();  
        System.out.println(l.size());  
    }  
}
```

Implementation Differences and their Effect

```
Collection c1 = new LinkedList();  
Collection c2;  
if(some_condition){  
    c2 = new ArrayList();  
} else {  
    c2 = new Vector();  
}  
c2.add(null);    // CALL SITE
```