CS 213 – Software Methodology

Spring 2023

Sesh Venugopal

Feb 8

Interfaces - 1

Comparing for inequality in a library module

```
public class Searcher {
    ...
    public static<T> boolean
    binarySearch(T[] list, T target) {
        ...
        list[index].___?___target

        ...
    }
    ...
}
```

Note the syntax used for a static method that accepts generically typed parameters

How to compare for inequality? All we know Is `T` is some `Object`, but `Object` does not define an inequality comparison method

Want to somehow specify that Ts are not *any* objects, but only those objects that have a *known* inequality comparison method

AND, this specification MUST be checkable by compiler, so that (a) our `binarySearch` will compile, and
(b) the client code's call to this method will be guaranteed to send in required type of object (with the known comparison method)

How to specify T type with method to check for inequality?

How to specify a T type with inequality method?

```
public class Searcher {
   ...
   public static<T> boolean
   binarySearch(T[] list, T target) {
       ...
       list[index].___?___target

       ...
   }
   ...
}
```

A class is a user-defined type, e.g. `Point` and `ColoredPoint` are types introduced by the program, which can be checked by the compiler (and appropriately matched at run time)

But we (library designer) can't *implement logic* for a new type instead of T that has, say, a `compareTo` method because the logic depends on the actual type of object that matches the generic type T: different matching objects would need different `compareTo` implementations (e.g. comparing strings is different from comparing points)

How to specify a T type with inequality method?

```
public class Searcher {
    ...
    public static<T> boolean
    binarySearch(T[] list, T target) {
        ...
        list[index].___?___target

        ...
    }
    ...
}
```

Solution is to make *like* we are defining a new class (type), with an inequality method, but stop short of actually implementing the method body – this is an INTERFACE

e.g. `java.lang.Comparable` interface, which defines a `compareTo` method, without a body:

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

Generic

No curly braces!!

Object of type T is compared

Then it's up to the client to implement a matching class and fill in the `compareTo` body as appropriate (`String` class implements this interface)

What interface to use with our binary search method?

What interface to use for T in binarySearch method?

```
public class Searcher {
   ...
   public static<T> boolean
   binarySearch(T[] list, T target) {
       ...
       list[index].___?___target

       ...
   }
   ...
```

We have the option of using any of the interfaces defined in Java, or roll our own if none of those fits our need

In our Searcher example, Comparable would be a perfect fit

```
public class Searcher {
   ...
   public static <Comparable<T>> boolean
   binarySearch(Comparable<T>[] list, Comparable<T> target) {
       ...
       list[index].compareTo(target)
       ...
   }
   ...
```

WILL NOT COMPILE
(not proper generic type syntax)