## **Generics and Collections**

COMP 2210 - Dr. Hendrix



SAMUEL GINN COLLEGE OF ENGINEERING

## **Generic types**

Generics allow a type variable to be used in place of a specific type name.

A type variable can be used to parameterize a class, interface, or method with respect to the types involved.

This allows classes, interfaces, and methods to deal with objects of different types at runtime while maintaining compile-time type safety.

```
public <T> int search(T[] a, T target)
  int i = 0;
  while ((i < a.length) && ( !a[i].equals(target) ))
  if (i < a.length)</pre>
     return i;
  else
     return -1;
}
```

ORACLE Tutorial <a href="http://download.oracle.com/javase/tutorial/java/generics/index.html">http://download.oracle.com/javase/tutorial/java/generics/index.html</a>

```
Generic type search example
   public static <T> int search(T[] a, T target) {
                                                                Assume that this is a static method in
      a class named SearchLib.
                                                                When calling this method, clients will
      if (i < a.length)
  return i;</pre>
                                                                preface the method name with the
                                                                class name.
          return -1;
   In a client ...
   String[] sarray = {"2", "4", "6", "8", "10"};
Integer[] iarray = {2, 4, 6, 8, 10};
Number[] narray = {2, 4, 6, 8, 10};
   Sample calls:
   SearchLib.<String>search(sarray, "8")
   SearchLib.<String>search(sarray, 8)
   SearchLib.search(sarray, 8)
   SearchLib.<Integer>search(iarray, 8)
   SearchLib.<Integer>search(narray, 8)
   SearchLib.<Number>search(narray, 8)
                                                                                     COMP 2210 • Dr. Hendrix • 3
```

```
Generic types and type safety

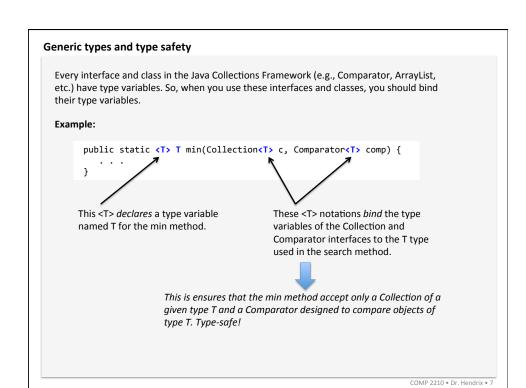
public class SearchLib {
   public static <T> int search(T[] a, T target) {
        int i = 0;
        while ((i < a.length) && (la[i].equals(target)))
        i++;
        if (i < a.length)
            return i;
        else
            return -1;
        }
   }
}

% javac -Xlint:unchecked SearchLib.java

No unchecked warnings, so this code is type-safe.
```

```
Generic types and type safety
   public class SearchLib {
       public static int search(Object[] a, Object target, Comparator c) {
           while ((i < a.length) && (c.compare(a[i], target) != 0))</pre>
          i++;
if (i < a.length)
              return i;
           else
              return -1:
      }
   % javac -Xlint:unchecked SearchLib.java
   SearchLib.java:5: warning: [unchecked] unchecked call to compare(T,T) as a member of the raw type Comparator
   while ((i < a.length) && (c.compare(a[i], target) != 0))
     where T is a type-variable:
T extends Object declared in interface Comparator
   1 warning
   %
   An unchecked warning, so this code is not type-safe. But where is the generic type variable?
   public interface Comparator<T> { . . . }
                                                                                                      COMP 2210 • Dr. Hendrix • 5
```

```
Generic types and type safety
  public class SearchLib {
      public static <MyType> int search(MyType[] a, MyType target, Comparator<MyType> c) {
         while ((i < a.length) && (c.compare(a[i], target) != 0))
         i++;
if (i < a.length)
                                                          Binds the Comparator's
                                                          type variable to
           return i;
                                                          МуТуре.
        else
           return -1;
                                    public interface Comparator<T> { . . . }
  }
  \% javac -Xlint:unchecked SearchLib.java \%
   No unchecked warnings, so this code is type-safe.
   Note: We could have used T for the type variable of the search method instead of
  MyType. I just didn't want to make you think that we had to use T because Comparator
   used T. It's completely up to you what you name the type variables.
```



## 

```
Bounded type parameters

public static <T extends Number> int search(T[] a, T target) {
    int i = 0;
    while ((i < a.length) && (!a[i].equals(target))) {
        i++;
    }
    if (i < a.length) return i;
    else
        return -1;
}

In a client ...

String[] sarray = {"2", "4", "6", "8", "10"};
Integer[] iarray = {2, 4, 6, 8, 10};
Number[] narray = {2, 4, 6, 8, 10};

Sample calls:

SearchLib.search(narray, 8)

SearchLib.search(iarray, 8)

SearchLib.<Number>search(narray, 8)

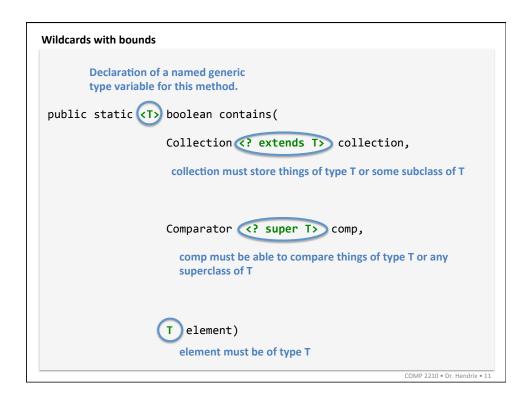
SearchLib.<Integer>search(narray, 8)

SearchLib.<Integer>search(narray, 8)

SearchLib.<Integer>search(narray, 8)

SearchLib.<Integer>search(sarray, "8")
```

## Wildcards ? The wildcard character represents an unnamed unknown type. Wildcards can be given upper bounds and lower bounds. <? extends MyType> MyType is an upper bound on this unknown type <? extends Book> An unknown type that is Book or any subclass of Book. <? super MyType> MyType is a lower bound on this unknown type <? super Book> An unknown type that is Book or any superclass of Book.



```
Generic types and type safety
    public static int search(Comparable[] a, Comparable target) {
         int i = 0;
while ((i < a.length) && (a[i].compareTo(target) != 0))</pre>
         i++;
if (i < a.length)
            return i;
         else
            return -1;
      }
                                                                                                Not
                                                                                                type-safe
      public static <T extends Comparable> int searchAlmostSafe(T[] a, T target) {
  int i = 0;
         while ((i < a.length) && (a[i].compareTo(target) != 0))</pre>
         i++;
if (i < a.length)
            return i;
         else
            return -1;
      }
      public static <T extends Comparable<? super T>> int searchSafe(T[] a, T target) {
         int i = 0:
         while ((i < a.length) && (a[i].compareTo(target) != 0))
         i++;
if (i < a.length)
            return i;
                                                                          Type-safe
            return -1;
```

