Week 6

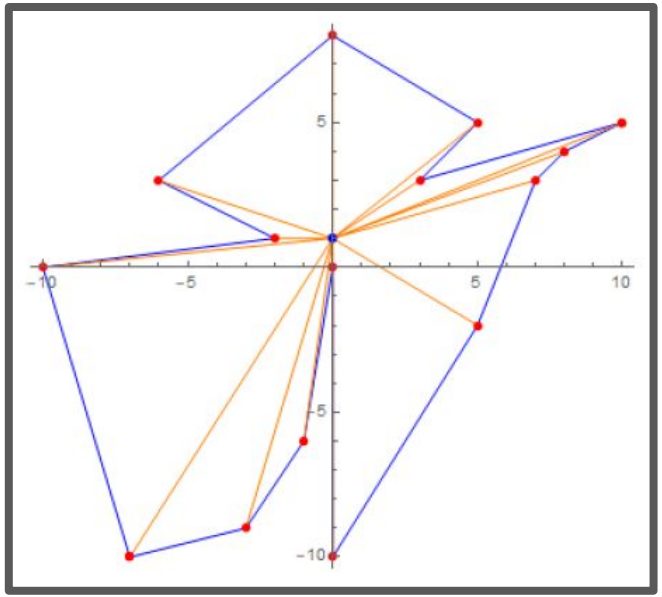
Generics

Project 2: Point Scanning

Due: March 3 (Sunday), midnight

Focuses on:

- Sorting algorithms
- Comparator interface
- Drawing geometry



Generics

```java
public int binarySearch(ArrayList<Integer> arr, Integer element)
{
    int low = 0;
    int high = arr.size() - 1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        if(element < arr.get(mid))
            high = mid - 1;

        if(element > arr.get(mid))
            low = mid + 1;

        if(element == arr.get(mid))
            return mid;
    }
    return -1;
}
```

A truly bad idea

```
public int binarySearch(ArrayList<Integer> arr, Integer element)

public int binarySearch(ArrayList<Double> arr, Double element)

public int binarySearch(ArrayList<String> arr, String element)

public int binarySearch(ArrayList<Point> arr, Point element)

public int binarySearch(ArrayList<Person> arr, Person element)
```

The answer is Generics!

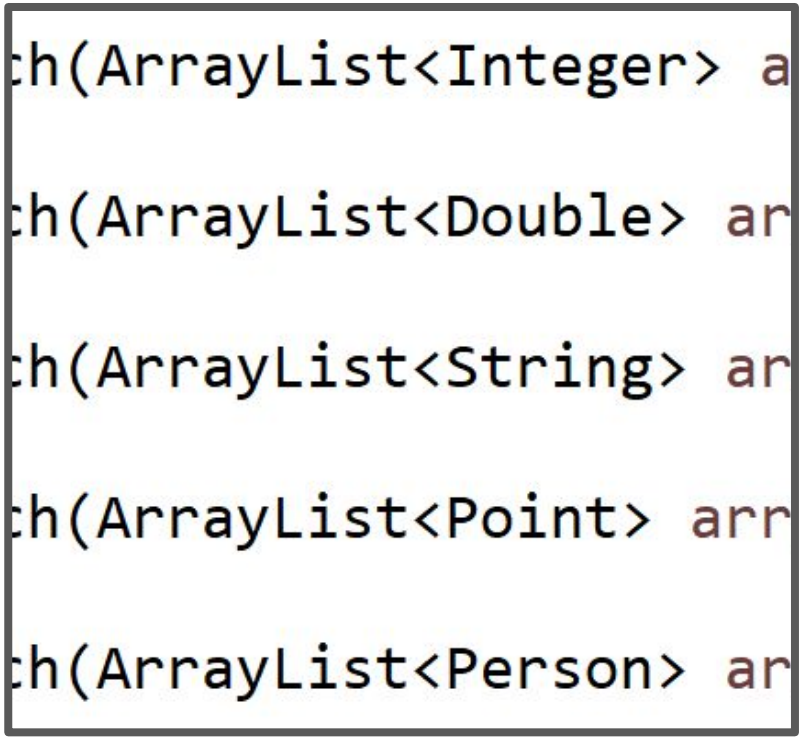
```
public <T> int binarySearch(ArrayList<T> arr, T element)
```

1 2 2

1. Define a generic type, T
2. Use T instead of a specific type to allow for multiple types to be used

You've been using Generics this whole time!

Everytime you use the <> brackets, you're defining the type of a generic for some class or interface



Using Generic types

```
public void binaryTest()
{
    ArrayList<Integer> arr1 = new ArrayList<>();
    ArrayList<String> arr2 = new ArrayList<>();
    ArrayList<Point> arr3 = new ArrayList<>();

    int i1 = binarySearch(arr1, 9);
    int i2 = binarySearch(arr2, "Mohammed");
    int i3 = binarySearch(arr3, new Point(0,5));
    int i4 = binarySearch(arr1, "Bad Idea");
}

public <T> int binarySearch(ArrayList<T> arr, T element)
```

Generics don’t work with primitives

```
public void binaryTest()
{
    int[] arr1 = new int[] {5, 7, 10, 17};
    double[] arr2 = new double[] {1.5, 4.3, 7.0};
    char[] arr3 = new char[] {'a', 'd', 'x', 'z'};

    int i1 = binarySearch(arr1, 9);
    int i2 = binarySearch(arr2, 4.3);
    int i3 = binarySearch(arr3, 'x');
}

public <T> int binarySearch(T[] arr, T element)
```

Java's solution in the Arrays class

static void	**sort**(char[] a) Sorts the specified array into ascending numerical order.
static void	**sort**(char[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	**sort**(double[] a) Sorts the specified array into ascending numerical order.
static void	**sort**(double[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	**sort**(float[] a) Sorts the specified array into ascending numerical order.
static void	**sort**(float[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	**sort**(int[] a) Sorts the specified array into ascending numerical order.
static void	**sort**(int[] a, int fromIndex, int toIndex) Sorts the specified range of the array into ascending order.
static void	**sort**(long[] a) Sorts the specified array into ascending numerical order.

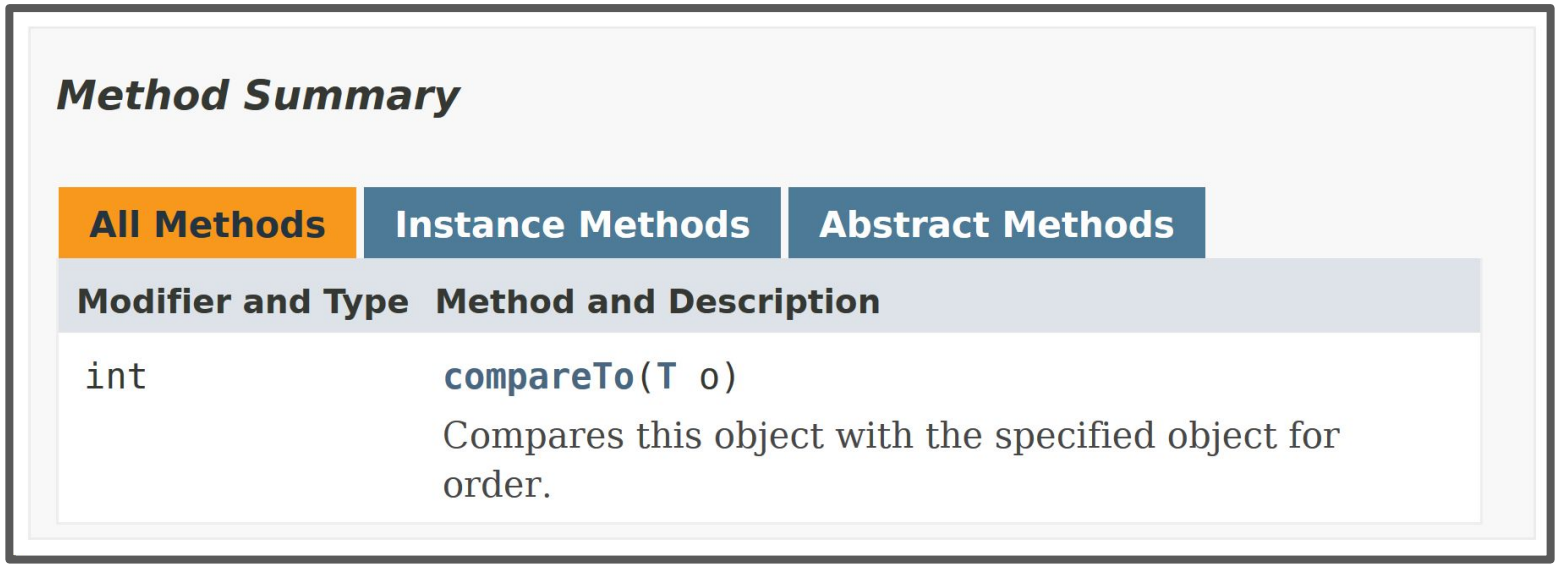
Sources: https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

We can’t use ‘<’ and ‘>’ on objects….

```
public <T> int binarySearch(ArrayList<T> arr, T element)
{
    int low = 0;
    int high = arr.size() - 1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        if(element < arr.get(mid))
            high = mid - 1;

        if(element > arr.get(mid))
            low = mid + 1;

        if(element == arr.get(mid))
            return mid;
    }
    return -1;
}
```

How do we compare objects?

We could use the Comparable interface!



Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
int	`compareTo(T o)` Compares this object with the specified object for order.

Sources: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Built-in Java Objects use Comparable

```
public final class Integer
extends Number
implements Comparable<Integer>
```

```
public final class Double
extends Number
implements Comparable<Double>
```

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

Sources:
https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
https://docs.oracle.com/javase/7/docs/api/java/lang/Double.html
https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html

Using the Comparable interface

```
Integer a = 6;
Integer b = 9;

if(a.compareTo(b) > 0)
{
    System.out.println("a is larger");
}
if(a.compareTo(b) < 0)
{
    System.out.println("b is larger");
}
if(a.compareTo(b) == 0)
{
    System.out.println("a is equal to b");
}
```

```java
public <T> int binarySearch(ArrayList<T> arr, T element)
{
    int low = 0;
    int high = arr.size() - 1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        if(arr.get(mid).compareTo(element) < 0)
            high = mid - 1;

        if(arr.get(mid).compareTo(element) > 0)
            low = mid + 1;

        if(arr.get(mid).compareTo(element) == 0)
            return mid;
    }
    return -1;
}
```

Bounding Generic types (Upper Bound)

```
public <T extends Comparable<T>>
int binarySearch(ArrayList<T> arr, T element)
```

A Comparable hierarchy

```
class Person implements Comparable<Person>
{
    protected int age;

    @Override
    public int compareTo(Person p) {
        return this.age - p.age;
    }
}
class Employee extends Person
class Boss extends Employee
```

```
class Person implements Comparable<Person>
class Employee extends Person
class Boss extends Employee

public void binaryTest()
{
    Employee e = new Employee();
    ArrayList<Employee> es = new ArrayList<>();
    binarySearch(es, e);
}

public <T extends Comparable<T>> int binarySearch(ArrayList<T> arr, T element)
```

```
class Person implements Comparable<Person>
class Employee extends Person
class Boss extends Employee

public void binaryTest()
{
    Employee e = new Employee();
    ArrayList<Employee> es = new ArrayList<>();
    binarySearch(es, e);
}

public <T extends Comparable<T>> int binarySearch(ArrayList<T> arr, T element)
```

Type of T: Employee Type of Comparable: Person

```
public void binaryTest()
{
    Person p = new Employee();
    ArrayList<Employee> es = new ArrayList<Employee>();
    ArrayList<Person> ps = es;
    binarySearch(ps, p);
}

public <T extends Comparable<T>> int binarySearch(ArrayList<T> arr, T element)
```

```
ArrayList<Employee> es = new ArrayList<Employee>();
ArrayList<Person> ps = es;
ps.add(new Person());
Employee e = es.get(0);
```

```
class Person                    implements Comparable<Person>
class Employee extends Person   implements Comparable<Employee>
class Boss extends Employee     implements Comparable<Boss>

public void binaryTest()
{
    Employee e = new Employee();
    ArrayList<Employee> es = new ArrayList<>();
    binarySearch(es, e);
}

public <T extends Comparable<T>> int binarySearch(ArrayList<T> arr, T element)
```

Why doesn’t this work?

Type Erasure

What is Type Erasure?

Compilers have 2 options when dealing with generic types:

```
class ArrayList<T>
{
    private T[] data;
    private int size;

    public T get(int index)
    public void add(T item)
    public T[] getAll()
}
```

What is Type Erasure?

Compilers have 2 options when dealing with generic types:

1. Generate new representation of the type for every new instance (C++)

```
class ArrayList<Integer>
{
    private Integer[] data;
    private int size;

    public Integer get(int index)
    public void add(Integer item)
    public Integer[] getAll()
}
```

```
class ArrayList<String>
{
    private String[] data;
    private int size;

    public String get(int index)
    public void add(String item)
    public String[] getAll()
}
```

Sources: http://www.angelikalanger.com/GenericsFAQ/FAQSections/TechnicalDetails.html#Topic3

What is Type Erasure?

Compilers have 2 options when dealing with generic types:

1. Generate new representation of the type for every new instance (C++)
2. Use only one representation with the most generalized types, then cast (Java)

```
class ArrayList
{
    private Object[] data;
    private int size;

    public Object get(int index)
    public void add(Object item)
    public Object[] getAll()
}
```

Sources: http://www.angelikalanger.com/GenericsFAQ/FAQSections/TechnicalDetails.html#Topic3

Before Type Erasure

```
ArrayList<Integer> arr = new ArrayList<>();
List<Integer> list = arr;
list.add(7);
list.add(95);
Integer data = list.get(1);
```

After Type Erasure

```
ArrayList arr = new ArrayList();
List list = arr;
list.add(7);
list.add(95);
Integer data = (Integer) list.get(1);
```

Bridge Methods

Before type erasure

```
interface Comparator<T>
{
    public int compare(T o1, T o2);
}

class PersonComparator implements Comparator<Person>
{
    public int compare(Person o1, Person o2)
    {
        return o1.age - o2.age;
    }
}
```

Bridge Methods

Before type erasure

```
interface Comparator<T>
{
    public int compare(T o1, T o2);
}

class PersonComparator implements Comparator<Person>
{
    public int compare(Person o1, Person o2)
    {
        return o1.age - o2.age;
    }
}
```

After type erasure

```
interface Comparator
{
    public int compare(Object o1, Object o2);
}

class PersonComparator implements Comparator
{
    public int compare(Person o1, Person o2)
    {
        return o1.age - o2.age;
    }

    //Bridge Method
    public int compare(Object o1, Object o2)
    {
        return compare((Person) o1, (Person) o2);
    }
}
```

Why can’t we implement Comparable twice?

```
class Person implements Comparable<Person>
{
    protected int age;

    //Comparable<Person> method and its erasure
    public int compareTo(Person p)
    public int compareTo(Object o)
}

class Employee extends Person implements Comparable<Employee>
{
    //Comparable<Employee> method and its erasure
    public int compareTo(Employee e)
    public int compareTo(Object o)
}
```

Back To Generics

How do we fix this then?

```
class Person implements Comparable<Person>
class Employee extends Person
class Boss extends Employee

public void binaryTest()
{
    Employee e = new Employee();
    ArrayList<Employee> es = new ArrayList<>();
    binarySearch(es, e);
}

public <T extends Comparable<T>> int binarySearch(ArrayList<T> arr, T element)
```

Wildcards + Lower Bounding

```
public <T extends Comparable<? super T>>
int binarySearch(ArrayList<T> arr, T element)
```

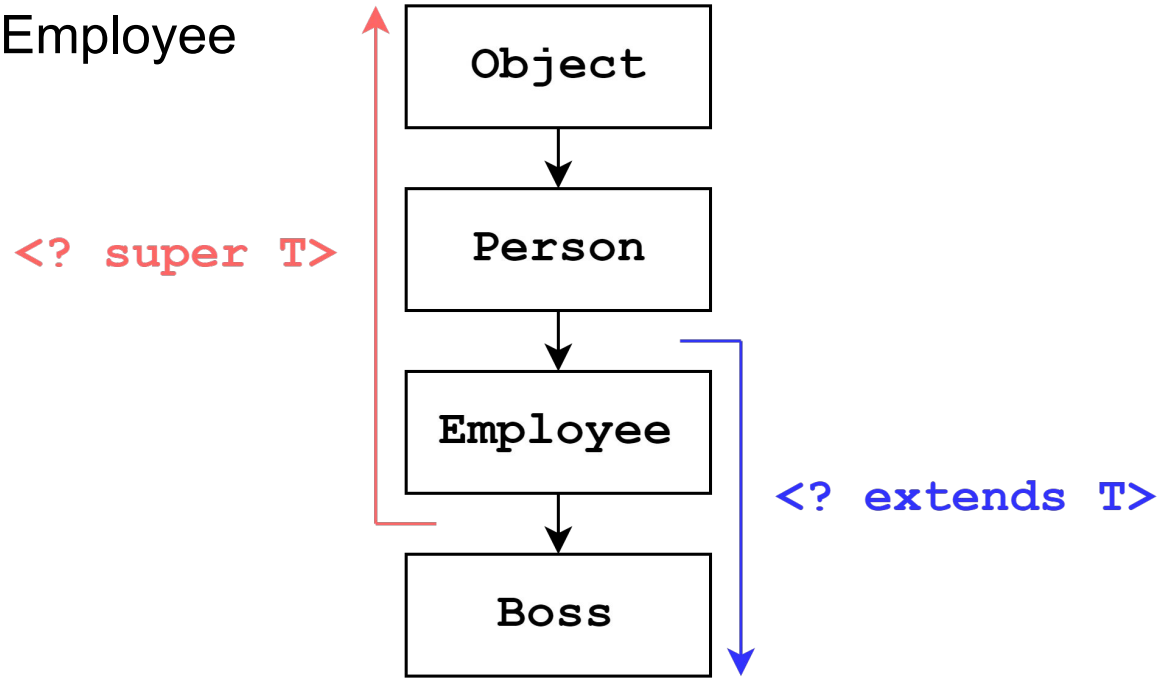
Wildcards (?)

```
public void printElements(List<?> arr)
{
    for(Object o : arr)
        System.out.println(o);
}
```

- Used to mean “Any type”
- Preferable if the type does not need to be reused

Type Bounding

Where T is Employee



What does it all mean?

```
<T extends Comparable<? super T>>
```

- Define a type, T
- T must implement the interface Comparable
- Comparable must be able to compare instances of T, or anything which is a superclass of T

```java
public <T extends Comparable<? super T>>
int binarySearch(ArrayList<T> arr, T element)
{
    int low = 0;
    int high = arr.size() - 1;

    while(low <= high)
    {
        int mid = (low + high) / 2;

        if(arr.get(mid).compareTo(element) < 0)
            high = mid - 1;

        if(arr.get(mid).compareTo(element) > 0)
            low = mid + 1;

        if(arr.get(mid).compareTo(element) == 0)
            return mid;
    }
    return -1;
}
```

End