CH. 2
OBJECT-ORIENTED PROGRAMMING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH DATA STRUCTURES AND ALGORITHMS IN JAVA, GOODRICH, TAMASSIA AND GOLDWASSER (WILEY 2016)
ADVANCED JAVA GENERICS
So far we learned to instantiate our generic objects like this:

- `List<Integer> l = new List<Integer>();`

However the Java compiler (with more recent versions) can infer the type parameters based on the first declaration

- `List<Integer> l = new List<>();`

  // Note that the <> are specified!
BOUNDDED TYPE PARAMETERS

• We can specify constraints on the generic parameters through interfaces to only accept certain types in generic classes/functions

• This extends the functionality of generic data to the methods of that interface

```java
public <T extends Comparable<T>> int compareTwo(T a, T b) {
    return a.compareTo(b); // Comparable<T> enforces a // having the compareTo function
}
```

• `extends` is always used, it generally refers to `extends` or `implements` in this context

• You can have multiple bounds as well:

```java
public class Foo<T extends A & B & C>
```
• As we have seen so far, type erasure means that our generic data was actually just `java.lang.Object` all along (with automatic casting).

• However, when we use bounded type parameters, we inform Java not to treat them as just `Object`, but as the bounded type.
INHERITANCE WITH GENERICS

• Thought question: Say class Foo extends class Bar

• Is a List<Foo> a subtype of List<Bar>?

• Actually no! It is important to know that inheritance is not applied to the type parameters, as in:
  A List<Foo> is not also a List<Bar>

• Still, as we have seen generic objects can subtype other generic objects, e.g., LeakyStack, ArrayList, etc.
• However, we can solve the subtype issue with wildcards

• In Java, in generic code a '?' is called a wildcard. It represents an unknown type, so List<Foo> and List<Bar> are subtypes of List<?> (but not List<Object>)

• Can be used to iterate over a generic list like so:
  ```java
  public static void print(List<?> list) {
    for (Object e : list) System.out.print(e + " ");
    System.out.println();
  }
  ```

• Note that print can be used with a list of any type of object
WILDCARDS

• You can additionally place bounds on wildcards using extends (for upper bounds) and super (for lower bounds)

• So we will see this:

  ```java
  public class Foo<T extends Comparable<? super T>>
  ```

  meaning Foo accepts any type that is comparable. Note that the comparable interface can be supported by any superclass of T
EXERCISE

• With your team. Create a generic interface called **Addable** with one function **add**

• Create two classes that implement **Addable**: **MyString** and **Point**.
  • **MyString** owns a single string and uses concatenation for adding
  • **Point** should be a 2D point in the Cartesian plane. Adding will be the sum of components.

• Create a generic function **sum** that sums an array of **Addable** objects

• Test your function in **main** with both random **MyStrings** and **Points**