# MySet

## 1 Introduction

Hello everyone! This week we'll write a class that fits into the Java Collections Framework by making our own implementation of the `Set` interface. This homework will cover `Collection` methods, generic typing, private inner classes, and `Iterator`s.

## 2 Problem Description

Your job is to write the class `MySet<E>` which implements the generic, that is, type-parameterized `Set<E>` interface. You must provide all of the methods that the `Set` interface requires. For a detailed description of what each of these methods is supposed to do, check the Java API page for the `Set` interface. Like last homework, when attempting to add to your `Set` whenever your backing array is full, your object should double the size of its backing array and copy the old elements over before adding the new element.

To help check to make sure your implementation is correct, we have provided the driver class `MySetTester.java`. You are not to modify this file - it should run correctly when provided with your `MySet` class. Your `MySet.java` file should contain javadoc comments for your `MySet` class, all public methods and, of course, pass all the checkstyle checks specified in `cs1331-checkstyle.xml`. Now, let's briefly go over the topics this homework covers.

## 3 Sets

Sets are simple: they are an aggregation of objects which can contain no duplicates. You can perform some operations on sets, such as adding and removing from them. You can ask a set if it contains a specific object, ask for an array containing all of the set elements in it, and perform a few other operations declared in `Collection`, such as `addAll(Collection)`, `retainAll(Collection)`, and `iterator()`. Your implementation of set will store its elements in a private array instance variable, and will add and remove from that backing array. For this assignment, you are not allowed to use `ArrayList`, since it already implements most of the methods you need for `MySet`.

# 4 Iterators

Iterators are objects that traverse collections and arrays. Collections that implement the `Iterable` interface can be the target of a for-each loop, which implicitly calls `iterator()` on the target collection and uses that iterator to update the loop variable. Your `MySet` class will implement the `Iterable` interface so that it can be the target of a for-each loop. Implementing `Iterable` means defining a single method: `iterator()`. Following good information hiding practices, your `iterator()` method should return an instance of a private inner class which implements the `Iterator` interface.

Private inner classes are classes defined within another class, and have access to private instance variables in their enclosing class. Because your custom `Iterator` will need to know the state of your backing array, you must make your `Iterator` an inner class instead of a separate class. You will need to define out the three methods the `Iterator` interface requires: `hasNext()`, `next()`, and `remove()`.

# 5 Hints

- You will be creating a backing array of a generic type. Java does not like making arrays of generic types directly, which is a limitation of generic types. Instead you must make an array of type Object and CAST it to an array of your generic type. This will cause the compiler to warn you about unsafe operations, so do not worry when you see that.
- Many of your methods, such as `addAll()` and `retainAll()`, can be simplified by calling other methods. Make sure you are duplicating as little code as possible. To `clear()`, rather than `remove()`ing each element, simply create a new backing array and reset the number of elements to zero.
- Some of the `Set` methods take in a parameter of a `Collection<?>`. The `?` means is that it can be a collection of any type. Some other methods take in a parameter of a `Collection<? extends E>`. This means that it can be a collection of elements of that type extends the element type of this `Set` instance. In either case, you can assume that elements in that `Set` extend from (and can be assigned to) an Object reference.

# 6 Solution Description

We give you the `MySetTester` class to use to test your code. This is the output it should print when run:

```
$ java MySetTester

Testing add:

Your size: 4, contains(Listen): true
Exp. size: 4, contains(Listen): true

Testing remove and removeAll:

Your size: 2, contains(Sorry): false
Exp. size: 2, contains(Sorry): false

Your size: 100
Exp. size: 100

Your size: 50
Exp. size: 50

ints should now only retain odd numbers 0 through 10

Testing your iterator:

1
3
5
7
9

Expected:

1
3
5
7
9

Yours:
Hey!
Listen!

Expected:
Hey!
Listen!

Clearing your set:

Your set is empty: true
Exp. set is empty: true
```

# 7 Turn-in Procedure

Submit all of the Java source files we provided as well as all new Java source files you created to T-Square. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft.

# 8 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight Friday. Do not wait until the last minute!