CS 5004 Module 4 Worksheet
Equality & Comparison

Each worksheet is to help you remember and understand the materials contained in each module so that you can get the most out of your learning experience. We start by asking questions that are covered in the module but will also ask you to apply this knowledge. By taking the time to answer each question carefully, these worksheets can serve as the basis for review for the more formal assessments in this class.

1. Define the difference between reflexivity, symmetry, and transitivity.

Reflexivity: A is equal to A (every object is equal to itself).

Symmetry: If A is equal to B, then B is equal to A.
Transitivity: If A is equal to B and B is equal to C, then A is equal to C.

2. For each of the following, indicate whether you should use `==` or `equals()` to determine if two variables of that type are equal to one another

`String`	equals()
`float`	equals()
`int`	==
`char`	==
`Book`	equals()
`Double`	equals()

3. The `Object` class is often called the mother of all parent classes because it contains methods that all classes must have. Name the three (3) that every class should consider overriding and indicate what they are used for.

- toString() method, this is used to convert the object to string.
- equals(object other), this is used for comparing two object.
- Hashcode(), this is used for returning the hashcode number for this object.

4. Consider the shapes example in Lesson 4.3. Implement a new class for this hierarchy that represents a triangle. In addition to the reference point that all shapes have, Triangle has 2 additional points that define the triangle's points. Ensure that you can determine equality of that it works using *double dispatch.*

```
public abstract class AbstractShape implements Shape {
  protected Point2D reference;

  protected boolean equalsCircle(Circle other) {
    return false; //by default "this" shape is not equal to a circle
  }

  protected boolean equalsRectangle(Rectangle other) {
    return false; //by default "this" shape is not equal to a rectangle
  }

  protected boolean equalsSquare(Square other) {
    return false; //by default "this" shape is not equal to a square
  }

protected boolean equalsTriangle(Triangle other) {
    return false; //by default "this" shape is not equal to a triangle
  }

}
```

```
Public class Triangle extends AbstractShape {
        Private Point2D left;
        Private Point2D right;
        Private Point2D top;

        public boolean equalsShape(Shape other) {
```

```
            if (other instanceof AbstractShape) {
                AbstractShape ashape = (AbstractShape) other;
                return ashape.equalsTriangle (this);
            }
            return false;
        }

        public boolean equalsTriangle(AbstractShape other) {
            if (other instanceof Triangle) {
                Triangle comp_triangle = (Triangle) other;
                return comp_triangle.getLeft() == this.getLeft()
                    && comp_triangle.getRight() == this.getRight()
                    && comp_triangle.getTop() == this.getTop();
            }
            return false;
        }


        Public Point2D getLeft() {
            return this.left;
        }

        Public Point2D getRight() {
            return this.right;
        }

        Public Point2D getTop() {
            return this.top;
        }

}
```

5. Consider the following class that represents a movie that we would like to be able to sort:

```
public class Movie {
  private int id;
  private int year;
  private String title;
  private String genre;
  private String director;
  private String leadrole;

  public Movie(int id, String title, String genre, int year,
               String director, String leadrole) {
    this.id = id;
    this.title = title;
    this.genre = genre;
    this.year = year;
    this.director = director;
    this.leadrole = leadrole;
  }
}
```

   a. Make the necessary changes to the class so that, by default, the movies can be sorted by their id.

```
public class Movie {
  private int id;
  private int year;
  private String title;
  private String genre;
  private String director;
  private String leadrole;
  private Movie[] movies = new Movie[10];
  private int size = 10;
  private int index = 0;

  public Movie(int id, String title, String genre, int year,
                          String director, String leadrole) {
              this.id = id;
              this.title = title;
              this.genre = genre;
              this.year = year;
              this.director = director;
              this.leadrole = leadrole;
              this.addMovie(movies, this);
            }
          }
```

```
Public void addMovie(Movie[] movies, Movie m) {
      //if the array is full, we need to resize the array
      if (index == size - 1) {
            //create a new array that has double size of the original array
            Movie[] newMoviesArray = new Movie[size * 2];

            //copy all the values in the original array to the new array
            For (int i = 0; i < size; i++) {
                  newMoviesArray[i] = movies[i];
            }
            //record the new size of the array
            Size *= 2;
            //reference the movie to the new movie array
            movies = newMoviesArray;
      }

      //if no need to resize the array or after the resize of the array
      movies[index] = m;
      //insert the m to the array, and increment the index by 1
      index++;
      sortMovies(m, index);
}

Public int getID() {
      Return this.id;
}

Public static void sortMovies(Movie[] movies, int size) {
      If (size <= 1) return;
      for (int k= 1; k < size; k++) {
            movie tempMovie = movies[k]
            int tempID = movies[k].getID();
            int j = k - 1;
            while ((j >= 0)
                  && (comparator.compare(movies[j].getID(), tempID) > 0)) {
                  movies[j+1] = movies[j];
                  j = j - 1;
            }
            movies[j+1] = tempMovie;
       }
}
```

b. Write code that can be used to determine ordering based on title without changing the default ordering.

```
public class Movie {
  private int id;
  private int year;
  private String title;
  private String genre;
  private String director;
  private String leadrole;
  private Movie[] movies = new Movie[10];
  private int size = 10;
  private int index = 0;

  public Movie(int id, String title, String genre, int year,
                          String director, String leadrole) {
              this.id = id;
              this.title = title;
              this.genre = genre;
              this.year = year;
              this.director = director;
              this.leadrole = leadrole;
              this.addMovie(movies, this);
            }
           }


Public void addMovie(Movie[] movies, Movie m) {
      //if the array is full, we need to resize the array
      if (index == size - 1) {
            //create a new array that has double size of the original array
            Movie[] newMoviesArray = new Movie[size * 2];

            //copy all the values in the original array to the new array
            For (int i = 0; i < size; i++) {
                  newMoviesArray[i] = movies[i];
            }
            //record the new size of the array
            Size *= 2;
            //reference the movie to the new movie array
            movies = newMoviesArray;
      }

      //if no need to resize the array or after the resize of the array
      movies[index] = m;
      //insert the m to the array, and increment the index by 1
```

```
      index++;
      sortMovies(m, index);
}

Public int getID() {
      Return this.id;
}

Public static void sortMovies(Movie[] movies, int size) {
      If (size <= 1) return;
      for (int k= 1; k < size; k++) {
            movie tempMovie = movies[k]
            int tempID = movies[k].getID();
            int j = k - 1;
            while ((j >= 0)
                  && (comparator.compare(movies[j].getID(), tempID) > 0)) {
                  movies[j+1] = movies[j];
                  j = j - 1;
            }
            movies[j+1] = tempMovie;
       }
}

Public String getTitle() {
      Return this.title;
}

Public static void sortArrayByTitle(Movie[] movies, int size) {
      If (size <= 1) return;
      for (int k= 1; k < size; k++) {
            movie tempMovie = movies[k]
            String tempTitle = movies[k].getTitle();
            int j = k - 1;
            while ((j >= 0)
                  && (comparator.compare(movies[j].getTitle(), tempTitle) > 0)) {
                  movies[j+1] = movies[j];
                  j = j - 1;
            }
            movies[j+1] = tempMovie;
       }

}
```

c. Write code that can be used to determining ordering based on the movie's director AND then the year the movie was published without changing the default ordering

```
Public static void sortArrayByDirector(Movie[] movies, int size) {
	If (size <= 1) return;
	for (int k= 1; k < size; k++) {
		movie tempMovie = movies[k]
		String tempDirector = movies[k].getDirector();
		int j = k - 1;
		while ((j >= 0)
			&& (comparator.compare(movies[j].getDirector(), tempDirector) > 0))
		{
			movies[j+1] = movies[j];
			j = j - 1;
		}
		movies[j+1] = tempMovie;
	 }

}

Public static void sortArrayByYear(Movie[] movies, int size) {
	If (size <= 1) return;
	for (int k= 1; k < size; k++) {
		movie tempMovie = movies[k]
		String tempYear = movies[k].getYear();
		int j = k - 1;
		while ((j >= 0)
			&& (comparator.compare(movies[j].getYear(), tempYear) > 0)) {
			movies[j+1] = movies[j];
			j = j - 1;
		}
		movies[j+1] = tempMovie;
	 }

}

Public String getDirector() {
	Return this.director;
}

Public int getYear() {
	Return this.year;
}
```

6. Consider the following class which can store a collection of integer values.

```
public class Bag {

  private int[] data;
  private int current;

  public Bag(int[] data) {
    this.data = data;
    for (int i = 0; i < data.length; i++) {
      this.data[i] = 0;
    }
    current = 0;
  }

  public void add(int element) {
    if (current < data.length) {
      data[current++] = element;
    }
  }

  public int remove() throws IllegalStateException {
    if (current == 0) {
      throw new IllegalStateException();
    }
    int result = data[current];
    data[current] = 0;
    current--;
    return result;
  }
}
```

a. In the code above, indicate what changes would be necessary to make the bag be able to store a collection of any type.

The data array is not initiated. And an array need to have fixed size.

```
public class Bag {

  private int size;
  private int[] data
  private int current;

  public Bag(int[] data, int size) {
    this.data = new int[size];
    this.size = size;
    for (int i = 0; i < size; i++) {
```

```
      this.data[i] = 0;
    }
    this.current = 0;
  }

  public void add(int element) {
    if (current < data.length) {
      data[current++] = element;
    }
  }

  public int remove() throws IllegalStateException {
    if (current == 0) {
      throw new IllegalStateException();
    }
    int result = data[current];
    data[current] = 0;
    current--;
    return result;
  }
}
```

b. Write a code snippet that would be able to create an instance of the Bag class that can store String values. Add 4 different values to the Bag. Then remove and print 2 values.

```
public class Bag<T> {

  private int size;
  private T[] data
  private int current;

  public Bag(T[] data, int size) {
    this.data = new T[size];
    this.size = size;
    for (int i = 0; i < size; i++) {
      this.data[i] = (T)0;
    }
    this.current = 0;
  }

  public void add(T element) {
    if (current < data.length) {
      data[current++] = element;
    }
  }
```

```
  public T remove() throws IllegalStateException {
    if (current == 0) {
      throw new IllegalStateException();
    }
    T result = data[current];
    data[current] = (T)0;
    current--;
    return result;
  }
}
```

```
Bag bagString = new Bag(String[] data, 10);
bagString.add("this");
bagString.add("is");
bagString.add("the");
bagString.add("string bag");
String removeFirst = bagString.remove();
System.out.println(removeFirst);
String removeSecond = bagString.remove();
System.out.println(removeSecond);
```

7. Consider the following implementation of *bubble sort:*

```
static void bubbleSort(int arr[]) {
  int n = arr.length;
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        // swap arr[j+1] and arr[j]
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
```

   a. Rewrite bubble sort as a generic method that can sort an array of anything.

```
Public static <T> void bubbleSort(T arr[], Comparator<T> comparator) {
  int n = arr.length;
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (comparator.compare(arr[i], arr[j]) < 0)) {
        // swap arr[j+1] and arr[j]
        T temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
```

b. Write a main method that could use your generic method for sorting both an array of integers and an array of strings.

```
Public static <Integer> void bubbleSort(Integer arr[], Comparator<Integer>
comparator) {
  int n = arr.length;
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (comparator.compare(arr[i], arr[j]) < 0)) {
        // swap arr[j+1] and arr[j]
        Integer temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}

Public static <String> void bubbleSort(String arr[], Comparator<String> comparator)
{
  int n = arr.length;
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (comparator.compare(arr[i], arr[j]) < 0)) {
        // swap arr[j+1] and arr[j]
        String temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
```

8. Explain what the following generic type declaration means and why you might use it.

```
<T extends Comparable<T>>
```

Here Comparable <T> is a generic interface, and extends means inheritance from a class or an interface. <T extends Comparable<T>> means this class either "implement" this "generic interface" Comparable<T> or create an interface and extend to this "generic interface" Comparable<T>.