```
//Purpose: Create an object of type Set
//Caspian Peavyhouse
//CS101-02
//12-4-2014

/* Data Table for
Variable or Constant          Purpose
_____________________         ___________________
setArray                      int array holding all values
*/

public class Set
{

   int[]setArray;

   /* Algorithm for Set
      setArray <-- new int [0]
   */

   public Set()
   {
      setArray = new int[0];
   }//Set


   /* Algorithm for makeEmpty
      this.setArray <-- new int[0]
   */
   public void makeEmpty()
   {
      this.setArray = new int[0];
   }//makeEmpty


   /* Algorithm for isEmpty
      if setArray.size() == 0
         return true
      else
         return false
   */
   public boolean isEmpty()
   {
      if (this.size() == 0)
         return true;
      else
         return false;
   }//isEmpty


   /* Algorithm for add
      if ( !this.elementOf(int addThis))
         int [] tempArray <-- new int [this.size() + 1]

         for (int i = 0; i < this.size(); i++)
            tempArray[i] = this.setArray[i]
         this.setArray <-- tempArray
         this.setArray[size() - 1] <-- addThis

         for (int run = 0; run < size() - 1; run++)
            for (int j = 0; j < size() - 1; j++)

                     if (setArray[j] > setArray[j + 1])
                        int temp <-- setArray[j]
                        this.setArray[j] <-- this.setArray[j + 1]
                        this.setArray[j + 1] <-- temp
   */
   /* Data Table for add
```

```
Variable or Constant          Purpose
_____________________         ___________________
addThis                       int value being added
tempArray                     temporary int array for addition
temp                          Int value used during array copy
*/

public void add(int addThis)
{

   if (!(this.elementOf(addThis)))
   {
      int [] tempArray = new int [this.size() + 1];

      for (int i = 0; i < this.size(); i++)
      {
         tempArray[i] = this.setArray[i];
      }

      this.setArray = tempArray;
      this.setArray[this.size() - 1] = addThis;

      for (int run = 0; run < this.size() - 1; run++)
      {
         for (int j = 0; j < this.size() - 1; j++)
         {
            if (setArray[j] > setArray[j + 1])
            {
               int temp = setArray[j];
               this.setArray[j] = this.setArray[j + 1];
               this.setArray[j + 1] = temp;
            }
         }
      }
   }
}//add


/* Algorithm for remove
   if ( this.elementOf(removeThis))
      for (int i = 0; i < this.size() - 1; i++)
         if (this.setArray[i] > removeThis)
            this.setArray[i] <-- this.setArray[i + 1]

      int [] tempArray <-- new int [this.size() - 1]
      for (int j = 0; j < tempArray.length; j++)
         tempArray[i] <-- this.setArray[i]

      this.setArray <-- tempArray
*/
/* Data Table for remove
Variable or Constant          Purpose
_____________________         ___________________
removeThis                    Int value being removed
tempArray                     Temporary int array in removal
*/

public void remove(int removeThis)
{
   if ( this.elementOf(removeThis))
   {
      for (int i = 0; i < this.size() - 1; i++)
      {
         if (this.setArray[i] >= removeThis)
         {
            this.setArray[i] = this.setArray[i + 1];
         }
      }
```

```
      int [] tempArray = new int [this.size() - 1];
      for (int j = 0; j < tempArray.length; j++)
      {
         tempArray[j] = this.setArray[j];
      }

      this.setArray = tempArray;
   }
}//remove


/* Algorithm for elementOf
   for (int i = 0; i < this.size(); i++)
         if (this.setArray[i] == checkThis)
            return true
   return false
*/
/* Data Table for elementOf
Variable or Constant          Purpose
_____________________         ___________________
checkValue                    int value being checked
*/

public boolean elementOf(int checkThis)
{
   for (int i = 0; i < this.size(); i++)
   {
      if (this.setArray[i] == checkThis)
      {
         return true;
      }
   }

   return false;
}//elementOf


/* Algorithm for size
   return this.setArray.length
*/
public int size()
{
   return this.setArray.length;
}//size


/* Algorithm for union
   Set newSet <-- new Set()
   for (int i = 0; i < this.size(); i++)
      newSet.add(this.setArray[i])

   for (int j = 0; j < otherSet.size(); j++)
      newSet.add(otherSet.setArray[j])

   return newSet
*/
/* Data Table for union
Variable or Constant          Purpose
_____________________         ___________________
otherSet                      set that is combined with setArray
newSet                        modified set being returned
*/

public Set union(Set otherSet)
{
   Set newSet = new Set();
   for (int i = 0; i < this.size(); i++)
```

```
   {
      newSet.add(this.setArray[i]);
   }

   for (int j = 0; j < otherSet.size(); j++)
   {
      newSet.add(otherSet.setArray[j]);
   }

   return newSet;

}//union


/* Algorithm for intersection
   Set newSet <-- new Set()
   int currentNum
   for (int i = 0; i < this.size(); i++)
      currentNum <-- this.setArray[i]
      if (otherSet.elementOf(currentNum))
         newSet.add(currentNum)
   return newSet
*/
/* Data Table for intersection
Variable or Constant          Purpose
_____________________         ___________________
otherSet                      set intersected with setArray
newSet                        modified set being returned
*/

public Set intersection(Set otherSet)
{
   Set newSet = new Set();
   int currentNum;
   for (int i = 0; i < this.size(); i++)
   {
      currentNum = this.setArray[i];
      if (otherSet.elementOf(currentNum))
      {
         newSet.add(currentNum);
      }
   }

   return newSet;
}//intersection


/* Algorithm for setDifference
   Set newSet <-- new Set()
   int currentNum
   for (int i = 0; i < this.size(); i++)
      currentNum <-- this.setArray[i]
      if  !(otherSet.elementOf(currentNum))
         newSet.add(currentNum)
   return newSet
*/
/* Data Table for setDifference
Variable or Constant          Purpose
_____________________         ___________________
otherSet                      set being tested against
newSet                        new set being returned
currentNum                    current int value being tested
*/

public Set setDifference(Set otherSet)
{
   Set newSet = new Set();
   int currentNum;
```

```
      for (int i = 0; i < this.size(); i++)
      {
         currentNum = this.setArray[i];
         if (!(otherSet.elementOf(currentNum)))
         {
            newSet.add(currentNum);
         }
      }

      return newSet;
   }//setDifference


   /* Algorithm for toString
      String output <-- new String("{")
      for (int i = 0; i < this.size(); i++)
         if (i == this.size() -1)
            output += "" + this.setArray[i]
         else
            output += "" + this.setArray[i] + ", "
      output += "}"
      return output
   */
   /* Data Table for toString
   Variable or Constant          Purpose
   _____________________         ___________________
   output                        String value being returned
   */

   public String toString()
   {
      String output = new String("{");
      for (int i = 0; i < this.size(); i++)
      {
         if (i == this.size() -1)
         {
            output += "" + this.setArray[i];
         }
         else
         {
            output += "" + this.setArray[i] + ", ";
         }
      }
      output += "}";

      return output;

   }//toString

}//Set
```