```
//Purpose: Driver(Main) class for the Set Class
//Caspian Peavyhouse
//CS101-02
//12-4-2014



import java.util.*;
import java.io.*;

public class SetDriver
{

/* Algorithm for main()

Set [] mainArray <-- new Set [100]


File inputFile <-- new File(args[0])
Scanner input <-- new Scanner(inputFile)

File outputFile <-- new File(args[1])
FileWriter writerOutput <-- new FileWriter(outputFile)
String currentLine <-- new String(input.nextLine())

writerOutput.write("SetDriver Version 4\n")
writerOutput.write("Written by Caspian Peavyhouse\n")
writerOutput.write("CS101-02\n\n")

do
      Scanner stringScan <-- new Scanner(currentLine)
      String currentString <-- new String(stringScan.next())

      switch (currentString)
      {
         case "C":
         {
            int setNum <-- stringScan.nextInt()
            if (setNum >= 0 && setNum <= 99)
            {
               callC(mainArray, setNum)
               writerOutput.write("Set number " + setNum +
                     " has been constructed and is empty\n")
            }
         } break

         case "I":
         {
            int setNum <-- stringScan.nextInt()
            String output <-- new String("")
            if (setNum >= 0 && setNum <= 99)
            {
               output <-- callI(mainArray, setNum)
               writerOutput.write(output + "\n")
            }
         } break

         case "S":
         {
            int setNum <-- stringScan.nextInt()
            String output <-- new String("")
            if (setNum >= 0 && setNum <= 99)
            {
               output <-- callS(mainArray, setNum)
               writerOutput.write(output + "\n")
            }
         } break
```

```
case "X":
{
   int setNum <-- stringScan.nextInt()

   String output <-- new String("")
   if (setNum >= 0 && setNum <= 99)
   {
      output <-- callX(mainArray, setNum)
      writerOutput.write(output + "\n")
   }
} break

case "A":
{
   int setNum <-- stringScan.nextInt()
   int newValue <-- stringScan.nextInt()

   if (mainArray[setNum] == null)
   {
      String output <-- ("There is no set number " + setNum)
      writerOutput.write(output + "\n")
   }
   else if (setNum >= 0 && setNum <= 99)
      callA(mainArray, setNum, newValue)
} break

case "R":
{
   int setNum <-- stringScan.nextInt()
   int newValue <-- stringScan.nextInt()
   if (mainArray[setNum] == null)
   {
      String output <-- ("There is no set number " + setNum)
      writerOutput.write(output + "\n")
   }
   else if (setNum >= 0 && setNum <= 99)
      callR(mainArray, setNum, newValue)
} break

case "F":
{
   int setNum <-- stringScan.nextInt()
   int checkValue <-- stringScan.nextInt()
   if (mainArray[setNum] == null)
   {
      String output <-- ("There is no set number " + setNum)
      writerOutput.write(output + "\n")
   }
   else
   {
      String output <-- callF(mainArray, setNum, checkValue)
      writerOutput.write(output + "\n")
   }
} break

case "U":
{
   int setOne <-- stringScan.nextInt()
   int setTwo <-- stringScan.nextInt()
   int targetSet <-- stringScan.nextInt()
   if (mainArray[setOne] == null || mainArray[setTwo] == null)
   {
      String output <-- ("Cannot take union
                          due to nonexisting set")
      writerOutput.write(output + "\n")
   }

   else
```

```
            callU(mainArray, setOne, setTwo, targetSet)
      } break

      case "N":
      {
         int setOne <-- stringScan.nextInt()
         int setTwo <-- stringScan.nextInt()
         int targetSet <-- stringScan.nextInt()
         if (mainArray[setOne] == null || mainArray[setTwo] == null)
         {
            String output <-- ("Cannot take intersection
                                due to nonexisting set")
            writerOutput.write(output + "\n")
         }

         else
            callN(mainArray, setOne, setTwo, targetSet)
      } break

      case "D":
      {
         int setOne <-- stringScan.nextInt()
         int setTwo <-- stringScan.nextInt()
         int targetSet <-- stringScan.nextInt()
         if (mainArray[setOne] == null || mainArray[setTwo] == null)
         {
            String output <-- ("Cannot take set difference
                               due to nonexisting set")
            writerOutput.write(output + "\n")
         }

         else
            callD(mainArray, setOne, setTwo, targetSet)
      } break

      case "P":
      {
         int setNum <-- stringScan.nextInt()
         if (mainArray[setNum] == null)
         {
            writerOutput.write("There is no set to print\n")
         }
         else
         {
            writerOutput.write(mainArray[setNum].toString() + "\n")
         }
      } break

      case "M":
      {
   int setNum <-- stringScan.nextInt()
   if (mainArray[setNum] != null)
   callM(mainArray, currentLine)
            } break

            default:
                  break

         }

         if (lastLine == true)
               break

         if (input.hasNextLine())
               currentLine <-- input.nextLine()
         else
               lastLine <-- true
```

```
      }while (true)

      writerOutput.close()
*/
/* Data Table for main
Variable or Constant       Purpose
_____________________      ___________________
inputFile                  argument for input file
outputFile                 argument for output file
currentLine                String value of line being evaluated
setNum                     Int value of the set being modified
output                     String value put into the writer
newValue                   Int value input  for add/remove
checkValue                 Int value input for elementOf
*/
   public static void main(String [] args)throws Exception
   {
      Set [] mainArray = new Set [100];


      File inputFile = new File(args[0]);
      Scanner input = new Scanner(inputFile);

      File outputFile = new File(args[1]);
      FileWriter writerOutput = new FileWriter(outputFile);
      String currentLine = new String(input.nextLine());

      writerOutput.write("SetDriver Version 4\n");
      writerOutput.write("Written by Caspian Peavyhouse\n");
      writerOutput.write("CS101-02\n\n");

      do
      {
         Scanner stringScan = new Scanner(currentLine);
         String currentString = new String(stringScan.next());

         switch (currentString)
         {
            case "C":
            {
               int setNum = stringScan.nextInt();
               if (setNum >= 0 && setNum <= 99)
               {
                  callC(mainArray, setNum);
                  writerOutput.write("Set number " + setNum +
                        " has been constructed and is empty\n");
               }
            } break;

            case "I":
            {
               int setNum = stringScan.nextInt();
               String output = new String("");
               if (setNum >= 0 && setNum <= 99)
               {
                  output = callI(mainArray, setNum);
                  writerOutput.write(output + "\n");
               }
            } break;

            case "S":
            {
               int setNum = stringScan.nextInt();
               String output = new String("");
               if (setNum >= 0 && setNum <= 99)
               {
                  output = callS(mainArray, setNum);
```

```
      writerOutput.write(output + "\n");
   }
} break;

case "X":
{
   int setNum = stringScan.nextInt();

   String output = new String("");
   if (setNum >= 0 && setNum <= 99)
   {
      output = callX(mainArray, setNum);
      writerOutput.write(output + "\n");
   }
} break;

case "A":
{
   int setNum = stringScan.nextInt();
   int newValue = stringScan.nextInt();

   if (mainArray[setNum] == null)
   {
      String output = ("There is no set number " + setNum);
      writerOutput.write(output + "\n");
   }
   else if (setNum >= 0 && setNum <= 99)
      callA(mainArray, setNum, newValue);
} break;

case "R":
{
   int setNum = stringScan.nextInt();
   int newValue = stringScan.nextInt();
   if (mainArray[setNum] == null)
   {
      String output = ("There is no set number " + setNum);
      writerOutput.write(output + "\n");
   }
   else if (setNum >= 0 && setNum <= 99)
      callR(mainArray, setNum, newValue);
} break;

case "F":
{
   int setNum = stringScan.nextInt();
   int checkValue = stringScan.nextInt();
   if (mainArray[setNum] == null)
   {
      String output = ("There is no set number " + setNum);
      writerOutput.write(output + "\n");
   }
   else
   {
      String output = callF(mainArray, setNum, checkValue);
      writerOutput.write(output + "\n");
   }
} break;

case "U":
{
   int setOne = stringScan.nextInt();
   int setTwo = stringScan.nextInt();
   int targetSet = stringScan.nextInt();
   if (mainArray[setOne] == null || mainArray[setTwo] == null)
   {
      String output = ("Cannot take union due to nonexisting set");
      writerOutput.write(output + "\n");
```

```
      }

      else
         callU(mainArray, setOne, setTwo, targetSet);
   } break;

   case "N":
   {
      int setOne = stringScan.nextInt();
      int setTwo = stringScan.nextInt();
      int targetSet = stringScan.nextInt();
      if (mainArray[setOne] == null || mainArray[setTwo] == null)
      {
         String output = ("Cannot take intersection due to nonexisting set");
         writerOutput.write(output + "\n");
      }

      else
         callN(mainArray, setOne, setTwo, targetSet);
   } break;

   case "D":
   {
      int setOne = stringScan.nextInt();
      int setTwo = stringScan.nextInt();
      int targetSet = stringScan.nextInt();
      if (mainArray[setOne] == null || mainArray[setTwo] == null)
      {
         String output = ("Cannot take set difference due to nonexisting set");
         writerOutput.write(output + "\n");
      }

      else
         callD(mainArray, setOne, setTwo, targetSet);
   } break;

   case "P":
   {
      int setNum = stringScan.nextInt();
      if (mainArray[setNum] == null)
      {
         writerOutput.write("There is no set to print\n");
      }
      else
      {
         writerOutput.write(mainArray[setNum].toString() + "\n");
      }
   } break;

   case "M":
   {
      int setNum = stringScan.nextInt();
      if (setNum >=0 && setNum <= 99)
         callM(mainArray, currentLine);
   } break;

   default:
      break;

}


if (input.hasNextLine())
   currentLine = input.nextLine();
else
   break;
```

```
   }while (true);



   writerOutput.close();
}//main

/* Algorithm for callC
       mainArray[setNum] = new Set();
*/
/* Data Table for callC
Variable or Constant    Purpose
____________________    ___________________
output                  String value being returned
setNum                  Int number of the set being changed
*/

public static void callC(Set [] mainArray, int setNum)
{
   mainArray[setNum] = new Set();
}//C

/* Algorithm for callI
   String output <-- new String("")
   if (mainArray[setNum] == null)
      output <-- ("There is no set number " + setNum)
   else if(mainArray[setNum].isEmpty())
      output <-- ("Set " + setNum + " is empty")
   else
      output <-- ("Set " + setNum + " is not empty")

   return output

*/
/* Data Table for callI
Variable or Constant    Purpose
____________________    ___________________
output                  String value being returned
setNum                  Int number of the set being changed
*/

public static String callI(Set [] mainArray, int setNum)
{
   String output = new String("");
   if (mainArray[setNum] == null)
       output = ("There is no set number " + setNum);
   else if(mainArray[setNum].isEmpty())
      output = ("Set " + setNum + " is empty");
   else
      output = ("Set " + setNum + " is not empty");

   return output;

}//callI

/* Algoritm for callS
   String output <-- new String("")
   if (mainArray[setNum] == null)
       output <-- ("There is no set number " + setNum)
   else
      output <-- ("Set " + setNum + " contains " + mainArray[setNum].size() + " elements")

   return output
*/
/* Data Table for callS
Variable or Constant    Purpose
____________________    ___________________
```

```
output                  String value being returned
setNum                  Int number of the set being changed
*/

public static String callS(Set [] mainArray, int setNum)
{
   String output = new String("");
   if (mainArray[setNum] == null)
       output = ("There is no set number " + setNum);
   else
      output = ("Set " + setNum + " contains "
               + mainArray[setNum].size() + " elements");

   return output;
}//callS


/* Algorithm for callX
   String output <-- new String("")
   if (mainArray[setNum] == null)
       output <-- ("There is no set number " + setNum + " to empty")
   else
      mainArray[setNum].makeEmpty()
      output <-- ("Set " + setNum + " has been emptied")

   return output
*/
/* Data Table for callX
Variable or Constant    Purpose
____________________    ___________________
output                  String value being returned
setNum                  Int number of the set being changed
*/
public static String callX(Set [] mainArray, int setNum)
{
   String output = new String("");
   if (mainArray[setNum] == null)
       output = ("There is no set number " + setNum + " to empty");
   else
   {
      mainArray[setNum].makeEmpty();
      output = ("Set " + setNum + " has been emptied");
   }

   return output;
}//callX


/* Algorithm for callA
   mainArray[setNum].add(newValue)

*/
/* Data Table for callA
Variable or Constant    Purpose
____________________    ___________________
newValue                Int number being added
setNum                  Int number of the set being changed
*/
public static void callA(Set [] mainArray, int setNum, int newValue)
{
   mainArray[setNum].add(newValue);
}//callA


/* Algorithm for callR
   mainArray[setNum].remove(newValue)

*/
```

```
/* Data Table for callR
Variable or Constant    Purpose
____________________    ___________________
newValue                Int number being removed
setNum                  Int number of the set being changed
*/
public static void callR(Set [] mainArray, int setNum, int newValue)
{
   mainArray[setNum].remove(newValue);
}//callR


/* Algorithm for callF
   String output <-- new String("")
   if (mainArray[setNum].elementOf(checkValue))
   {
      output <-- (checkValue + " is in the set")
   }
   else
   {
      output <-- (checkValue + " is not in the set")
   }

   return output
*/
/* Data Table for callF
Variable or Constant    Purpose
____________________    ___________________
checkValue              Int number being checked
setNum                  Int number of the set being changed
output                  String value bing returnd
*/
public static String callF(Set [] mainArray, int setNum, int checkValue)
{
   String output = new String("");
   if (mainArray[setNum].elementOf(checkValue))
   {
      output = (checkValue + " is in the set");
   }
   else
   {
      output = (checkValue + " is not in the set");
   }

   return output;
}//callF


/* Algorithm for callU
   mainArray[targetSet] <-- mainArray[setOne].union(mainArray[setTwo])
*/
/* Data Table for callU
Variable or Constant    Purpose
____________________    ___________________
setOne                  Int number of first set
setTwo                  Int number of second set
targetSet               Int number of set being modified
*/

public static void callU(Set [] mainArray, int setOne,
                              int setTwo, int targetSet)
{
   mainArray[targetSet] = mainArray[setOne].union(mainArray[setTwo]);
}//callU


/* Algorithm for callN
   mainArray[targetSet] <-- mainArray[setOne].intersection(mainArray[setTwo])
```

```
*/
/* Data Table for callN
Variable or Constant    Purpose
____________________    ___________________
setOne                  Int number of first set
setTwo                  Int number of second set
targetSet               Int number of set being modified
*/
public static void callN(Set [] mainArray, int setOne,
                              int setTwo, int targetSet)
{
   mainArray[targetSet] = mainArray[setOne].intersection(mainArray[setTwo]);
}//callN


/* Algorithm for callD
   mainArray[targetSet]= mainArray[setOne].setDifference(mainArray[setTwo])

*/
/* Data Table for callD
Variable or Constant    Purpose
____________________    ___________________
setOne                  Int number of first set
setTwo                  Int number of second set
targetSet               Int number of set being modified
*/
public static void callD(Set [] mainArray, int setOne,
                              int setTwo, int targetSet)
{
   mainArray[targetSet]= mainArray[setOne].setDifference(mainArray[setTwo]);
}//callD


/* Algorithm for callM
   Scanner lineScan <-- new Scanner(currentLine)
   lineScan.next()//to pass the M
   int setNum <-- lineScan.nextInt()
   callC(mainArray, setNum)
   int currentNum <-- lineScan.nextInt()
   while (true)
   {
      mainArray[setNum].add(currentNum)

      if (!lineScan.hasNext())
      {
         break
      }
      currentNum <-- lineScan.nextInt()
   }

*/
/* Data Table for callM
Variable or Constant    Purpose
____________________    ___________________
currentLine             String value of line being read
setNum                  Int number of set being modified
currentNum              Int number currently being read
*/
public static void callM(Set [] mainArray, String currentLine)
{
   Scanner lineScan = new Scanner(currentLine);
   lineScan.next();//to pass the M
   int setNum = lineScan.nextInt();
   callC(mainArray, setNum);
   int currentNum = lineScan.nextInt();
   while (true)
   {
```

```
         mainArray[setNum].add(currentNum);

         if (!lineScan.hasNext())
         {
            break;
         }
         currentNum = lineScan.nextInt();
      }
   }//callM



}//SetDriver
```