Writen by Caspian Peavyhouse
CS101-02
12/8/2014

| (+) SetDriver |
| --- |
| |
| (+) main()<br>(+) callC(mainArray:Set [], setNum: int)<br>(+) callI(mainArray:Set [], setNum: int):String<br>(+) callS(mainArray:Set [], setNum: int):String<br>(+) callX(mainArray:Set [], setNum: int):String<br>(+) callA(mainArray:Set [], setNum: int, newValue:Int)<br>(+) callR(mainArray:Set [], setNum: int, newValue:Int)<br>(+)callF(mainArray:Set [], setNum: int, checkValue:Int):String<br>(+) callU(mainArray:Set [], seOne, setTwo,targetSet: int)<br>(+) callN(mainArray:Set [], seOne, setTwo,targetSet: int)<br>(+) callD(mainArray:Set [], seOne, setTwo,targetSet: int)<br>(+) callM((mainArray:Set [], currentLine:String) |

| Legend |
| --- |
| (+) public<br>(-) private<br>( ) package<br>(#) protected |

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 (input.hasNextLine())
            currentLine <-- input.nextLine()
        else
            break


    }while (true)

    writerOutput.close()


Algoritm for callC
        mainArray[setNum] <-- new Set()

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

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

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

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


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

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

Algorithm for callU
        mainArray[targetSet] <-- mainArray[setOne].union(mainArray[setTwo])


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

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


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()
```

```
boolean lastNum <-- false
 while (true)
        mainArray[setNum].add(currentNum)

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