UML Class Diagrams

Joseph Maples CS101
Design for an object-oriented implementation of sets

+	public
-	private
	package
#	protected

legend

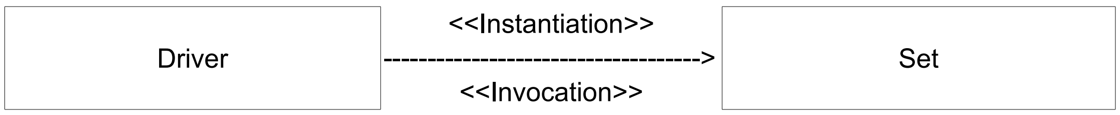
+	Driver
+	set:Set[]
+	printer:PrintStream
+	file:Scanner
+	main(args:String [])
+	nextCommand()
+	construct()
+	isEmpty()
+	size()
+	empty()
+	add()
+	remove()
+	find()
+	union()
+	intersect()
+	difference()
+	print()
+	create()
+	message()

+	Set
-	set:int[]
+	Set()
+	getSet():int[]
-	setSet()
+	makeEmpty()

- + isEmpty():boolean
- + add(add:int)
- + remove(rem:int)
- + elementOf(value:int):boolean
- + size():int
- + union(join:Set):Set
- + intersect(intersect:Set):Set
- + setDifference(diff:Set):Set
- + toString():String

UML Class Interaction Diagram

Joseph Maples CS101
Design for an object-oriented implementation of sets



Joseph Maples CS101
Design for an object-oriented implementation of sets

Data Table for Class Set

Variable or Constant	Type	Purpose
set	int[]	A set of numbers

Data Table for setSet(set:int[])

Variable or Constant	Type	Purpose
set	int[]	A set of numbers
set	int[]	A new set of numbers

Data Table for makeEmpty()

Variable or Constant	Type	Purpose
element	int	An index of the set

Data Table for add(add:int)

Variable or Constant	Type	Purpose
add	int	Number to add to set
added	int[]	The current set plus the added number
element	int	An index of the set

Data Table for remove(rem:int)

Variable or Constant	Type	Purpose
rem	int	Number to remove from set
removed	Set	Set that contains the current set without rem
element	int	An index of the set

Data Table for elementOf(value:int)

Variable or Constant	Type	Purpose
element	int	An index of the set
value	int	Value to check for in set

Data Table for union(join:Set)

Variable or Constant	Type	Purpose
newSet	Set	Set to add union to
join	Set	Set to join with current set
element	int	An index of the set

Data Table for intersection(intersect:Set)

Variable or Constant	Type	Purpose
newSet	Set	Set to add intersection to
intersect	Set	Set to intersect with current set
element	int	An index of the set

Data Table for setDifference(diff:Set)

Variable or Constant	Type	Purpose
newSet	Set	Set to add difference to

diff	Set	Set to find difference with current set
element	int	An index of the set

Data Table for toString()

Variable or Constant	Type	Purpose
setString	String	The current set as a string
element	int	An index of the set

Joseph Maples CS101
Design for an object-oriented implementation of sets

Data Table for Class Driver

Variable or Constant	Type	Purpose
sets	Set[]	The 100 sets the driver handles
file	Scanner	Scan the input file
printer	PrintStream	Write to the output file

Data table for main(args:String [])

Variable or Constant	Type	Purpose
inFile	File	input file
outFile	file	output file
file	Scanner	Scan the input file
printer	PrintStream	Write to the output file

Data Table for nextCommand()

Variable or Constant	Type	Purpose
first	String	The first string of a command
cmd	char	The first letter of the command string

Data Table for construct()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array
sets	Set[]	The 100 sets the driver handles

Data Table for isEmpty()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array

Data Table for size()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array

Data Table for empty()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array

Data Table for add()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array
element	int	Element inside the selected set

Data Table for remove()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array
element	int	Element inside the selected set

Data Table for find()

Data Table for Driver

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array
element	int	Element inside the selected set

Data Table for union()

Variable or Constant	Type	Purpose
set1	int	First set to join
set2	int	Second set to join
set3	int	Set to contain the union
sets	Set[]	The 100 sets the driver handles

Data Table for intersect()

Variable or Constant	Type	Purpose
set1	int	First set to intersect
set2	int	Second set to intersect
set3	int	Set to contain the intersection
sets	Set[]	The 100 sets the driver handles

Data Table for difference()

Variable or Constant	Type	Purpose
set1	int	First set to find difference
set2	int	Second set to find difference
set3	int	Set to contain the differences
sets	Set[]	The 100 sets the driver handles

Data Table for print()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array

Data Table for is create()

Variable or Constant	Type	Purpose
slot	int	Slot (index) in the sets array
sets	Set[]	The 100 sets the driver handles

Data Table for message(first:String)

Variable or Constant	Type	Purpose
first	String	First part of message

Joseph Maples CS101
Design for an object-oriented implementation of sets

Set Class Algorithms

```
Set()
    setSet(new int[0])

getSet()
    return set

setSet([] set)
    this.set equals set

makeEmpty()
    for element equals set.length - 1 loop till element is greater than or equal to 0 by element - 1
        remove(set[element])

isEmpty()
    if (set.length equals 0)
        return true
    return false

add(add)
    if (NOT elementOf(add))
        [] added equals [0..set.length + 1-1]
        for element equals 0 loop till element is less than set.length by element + 1 each step
            added[element] equals set[element]
        added[set.length] equals add
        setSet(added)
        added equals null

remove(rem)
    if (elementOf(rem))
        Set removed equals new Set()
        for element equals 0 loop till element is less than set.length - 1 by element + 1 each step
            if (set[element] does not equal rem)
                removed.add(set[element])
        setSet(removed.getSet())
        removed equals null

elementOf(value)
    for element equals 0 loop till element is less than set.length by element + 1 each step
        if (set[element] equals value)
            return true
    return false

size()
    return set.length
```

```
union(Set join)
    Set newSet equals new Set()
    for element equals 0 loop till element is less than join.size() by element + 1 each step
        if (NOT elementOf(join.getSet()[element]))
            newSet.add(join.getSet()[element])
    for element equals 0 loop till element is less than set.length by element + 1 each step
        newSet.add(set[element])
    return newSet

intersection(Set intersect)
    Set newSet equals new Set()
    for element equals 0 loop till element is less than intersect.size() by element + 1 each step
        if (elementOf(intersect.getSet()[element]))
            newSet.add(intersect.getSet()[element])
    return newSet

setDifference(Set diff)
    Set newSet equals new Set()
    for element equals 0 loop till element is less than set.length by element + 1 each step
        if (NOT diff.elementOf(set[element]))
            newSet.add(set[element])
    return newSet

toString()
    String setString equals "{"
    if (NOT isEmpty())
        setStriing += set[0]
    for element equals 1 loop till element is less than set.length by element + 1 each step
        setString += "," + set[element]
    setString += "}"
    return setString
```

each step

Joseph Maples CS101
Design for an object-oriented implementation of sets

Driver Class Algorithms

```
main(String[] args) throws Exception
    File inFile equals new File(args[0])
    File outFile equals new File(args[1])
    file equals new Scanner(inFile)
    printer equals new PrintStream(outFile)
    sets equals Set[0..98]
    while (file.hasNext())
        nextCommand()

nextCommand()
    String first equals file.next()
    cmd equals first.charAt(0)
    switch (cmd) :
        case 'C':
            construct()
            break
        case 'I':
            isEmpty()
            break
        case 'S':
            size()
            break
        case 'X':
            empty()
            break
        case 'A':
            add()
            break
        case 'R':
            remove()
            break
        case 'F':
            find()
            break
        case 'U':
            union()
            break
        case 'N':
            intersect()
            break
        case 'D':
            difference()
            break
        case 'P':
            print()
            break
         case 'M':
            create()
            break
         case '#':
            message(first)
            break
         default:
            print to file ("Invalid command!")


construct()
      slot equals file.nextInt()
      sets[slot] equals new Set()
      print to file ("Set " plus slot plus " has been constructed.")

isEmpty()
      slot equals file.nextInt()
      if (sets[slot] equals null)
         print to file ("Set " plus slot plus " does not exist!")
      else if (sets[slot].isEmpty())
         print to file ("Set " plus slot plus " is empty.")
      else
         print to file ("Set " plus slot plus " is not empty.")

size()
      slot equals file.nextInt()
      if (sets[slot] equals null)
         print to file ("Set " plus slot plus " does not exist, it has no size.")
      else if (sets[slot].isEmpty())
         print to file ("Set " plus slot plus " is empty.")
      else
         print to file ("Set " plus slot plus " is " plus sets[slot].size() plus " elements long.")

empty()
      slot equals file.nextInt()
      if (sets[slot] equals null)
         print to file ("Cannot empty set " plus slot plus " that doesn't exsit!")
      else
         sets[slot].makeEmpty()
         print to file ("Set " plus slot plus " has been emptied.")

add()
      slot equals file.nextInt()
      element equals file.nextInt()
      if (sets[slot] equals null)
         print to file ("No set to add value to!")
      else
```

```
          sets[slot].add(element)

remove()
        slot equals file.nextInt()
        element equals file.nextInt()
        if (sets[slot] equals null)
          print to file ("No set to remove value from!")
        else
          sets[slot].remove(element)

find()
        slot equals file.nextInt()
        element equals file.nextInt()
        if (sets[slot] equals null)
          print to file ("No set in which to find value.")
        else if (sets[slot].elementOf(element))
          print to file (element plus " is a part of set " plus slot)
        else
          print to file (element plus " is not a part of set " plus slot)

union()
        set1 equals file.nextInt()
        set2 equals file.nextInt()
        set3 equals file.nextInt()
        if (sets[set1] equals null OR sets[set2] equals null)
          print to file ("Cannot join a nonexistent set!")
        else
          sets[set3] equals sets[set1].union(sets[set2])

intersect()
        set1 equals file.nextInt()
        set2 equals file.nextInt()
        set3 equals file.nextInt()
        if (sets[set1] equals null OR sets[set2] equals null)
          print to file ("Cannot intersect a nonexistent set!")
        else
          sets[set3] equals sets[set1].intersection(sets[set2])

difference()
        set1 equals file.nextInt()
        set2 equals file.nextInt()
        set3 equals file.nextInt()
        if (sets[set1] equals null OR sets[set2] equals null)
          print to file ("Cannot find difference a nonexistent set!")
        else
          sets[set3] equals sets[set1].setDifference(sets[set2])

print()
        slot equals file.nextInt()
        if (sets[slot] equals null)
            print to file ("Cannot print set " plus slot plus " that does not exist!")
        else
            print to file (sets[slot].toString())

create()
        slot equals file.nextInt()
        sets[slot] equals new Set()
        while (file.hasNextInt())
            sets[slot].add(file.nextInt())

message(String first)
        print to file (first plus file.nextLine())
```