package lab03;

/\*\*

\* Generic interface that defines methods for use with a bag. The bag can hold

\* any data types, does not order the items and does not guard against

\* duplicate entries.

\* @param T Datatype of the bag

\* @author Jaden Young

\*/

public interface Bag<T> {

/\*\* Returns the current number of items in the bag \*/

public int getCurrentSize();

/\*\* Tells whether or not the bag is empty \*/

public boolean isEmpty();

/\*\* Tells whether or not the bag is full \*/

public boolean isFull();

/\*\*

\* Adds an item to the bag.

\* Returns true if the item was added successfully, false if it failed

\* @param item Item to be added to the bag

\*/

public boolean add(T item);

/\*\*

\* Removes a random item from the bag, if the bag is not empty.

\* Returns the item that was removed

\*/

public T remove();

/\*\*

\* Removes the first occurrence of the specified item in the bag.

\* Returns true if removal was successful, false if not.

\* @param item Item to be removed

\*/

public boolean remove(T item);

/\*\* Clears the bag, removing all items \*/

public void clear();

/\*\*

\* Returns the number of times a specified object appears in the bag

\* @param item Item to be searched for

\*/

public int getFrequencyOf(T item);

/\*\*

\* Returns true if specified item is present in the bag, false if not

\* @param item Item to be checked for

\*/

public boolean contains(T item);

}

package lab03;

import java.util.Random;

/\*\*

\* Uses an array as a data structure for a generic bag.

\* @author Jaden Young

\*/

public class ArrayBag<T> implements Bag<T> {

private T[] bag;

private int count;

/\*\*

\* Default constructor<BR>

\* Creates an array of generic type with length 50

\*/

public ArrayBag() {

Object[] temp = new Object[50];

bag = (T[])temp;

temp = null;

count = 0;

}

/\*\*

\* Overloaded constructor<BR>

\* Creates an array of generic type with specified length

\* @param size Length of the array

\*/

public ArrayBag(int size) {

Object[] temp = new Object[size];

bag = (T[])temp;

temp = null;

count = 0;

}

/\*\* Returns the number of items in the bag \*/

@Override

public int getCurrentSize() {

return count;

}

/\*\* Returns true if the bag is empty, false if not \*/

@Override

public boolean isEmpty() {

return count == 0;

}

/\*\* Returns true if the bag is full, false if not \*/

@Override

public boolean isFull() {

return count == bag.length;

}

/\*\*

\* Adds an item to the end of the bag.<BR>

\* If the bag is full, the length is doubled, and the item is then added.

\* Returns true if the addition was successful, false if not

\* @param item Item to be added

\* @return True if the item was added successfully, false if not

\*/

@Override

public boolean add(T item) {

boolean success = false;

if(count == bag.length) {

Object[] temp = new Object[2 \* bag.length];

for(int i = 0; i < count; i++) {

temp[i] = bag[i];

}

bag = (T[])temp;

temp = null;

}

bag[count] = item;

count++;

return bag[count - 1] == item;

}

/\*\*

\* Removes a random item from the bag.<BR>

\* Returns the removed item

\* @return item removed from the bag

\*/

@Override

public T remove() {

Random rand = new Random();

int index = rand.nextInt(count);

T removed = bag[index];

for(int i = index; i < count; i++) {

bag[i] = bag[i + 1];

}

count--;

return removed;

}

/\*\*

\* Removes a specified item from the bag.<BR>

\* All following elements are shifted 1 to the left to fill the hole.<BR>

\* Returns true if item was removed successfully, false if not.

\* @param item Item to be removed

\* @return True if removal was successful, false if removal failed/item was

\* not found

\*/

@Override

public boolean remove(T item) {

for(int i = 0; i < count; i++) {

if(bag[i].equals(item)) {

for(int j = i; j < count; j++) {

bag[j] = bag[j + 1];

}

count--;

return true;

}

}

return false;

}

/\*\*

\* Removes all entries from the bag.<BR>

\* The method works by simply assigning the reference to a new bag,

\* resetting the length to 50.

\*/

@Override

public void clear() {

Object[] temp = new Object[50];

bag = (T[])temp;

temp = null;

count = 0;

}

/\*\*

\* Returns the number of times a specified item appears in the bag

\* @param item Item to be searched for

\* @return Number of times the item appears in the bag

\*/

@Override

public int getFrequencyOf(T item) {

int freq = 0;

for(int i = 0; i < count; i++) {

if(bag[i].equals(item))

freq++;

}

return freq;

}

/\*\*

\* Tells whether or not a specified item is in the bag

\* @param item Item to search for

\* @return True if item is found, false if not

\*/

@Override

public boolean contains(T item) {

for(int i = 0; i < count; i++) {

if(bag[i].equals(item))

return true;

}

return false;

}

/\*\*

\* Returns the item stored at the specified index position

\* @param index Index of the item that will be returned

\* @return item stored at the specified index

\*/

public T getItem(int index) throws ArrayIndexOutOfBoundsException {

if(index < 0 || index > count)

throw new ArrayIndexOutOfBoundsException("ERROR: There was a "

+ "problem retrieving the data. Specified index position "

+ "is empty.");

return bag[index];

}

/\*\*

\* Returns the current capacity of the bag (not the number of items stored)

\* @return bag capacity

\*/

public int getSize() {

return bag.length;

}

/\*\*

\* Returns a copy of the bag

\* @return A reference to a copy of the bag

\*/

public T[] getCopy() {

Object[] tempObj = new Object[bag.length];

for(int i = 0; i < count; i++) {

tempObj[i] = bag[i];

}

T[] tempT = (T[])tempObj;

tempObj = null;

return tempT;

}

/\*\*

\* toString method that returns a string comprised of a combination of

\* the toString methods for each element in the bag.

\* @return Concatenated String with the output of the toString method from

\* every element in the bag

\*/

@Override

public String toString() {

String output = "";

for(int i = 0; i < this.getCurrentSize(); i++) {

output += "\n" + bag[i].toString();

}

return output;

}

}

package lab03;

/\*\*

\* Defines a Player object with values for name, the position they play, and

\* their jersey number.

\* @author Jaden Young

\*/

public class Player {

//instance variables

private String name;

private String positionPlayed;

private int jerseyNumber;

/\*\*

\* Constructs a new Player object with values for all of the fields

\* @param xName Name of the player

\* @param xPosition Position of the player

\* @param xNumber The player's number

\*/

public Player(String xName, String xPosition, int xNumber){

name = xName;

positionPlayed = xPosition;

this.setJerseyNumber(xNumber);

}

//accessor methods---------------------------------

/\*\*

\* Returns the name of the player

\* @return Name of the player

\*/

public String getName() {

return name;

}

/\*\*

\* Returns the position of the player

\* @return Position of the player

\*/

public String getPositionPlayed() {

return positionPlayed;

}

/\*\*

\* Returns the number of the player

\* @return Player's number

\*/

public int getJerseyNumber() {

return jerseyNumber;

}

//mutator methods -------------------------

/\*\*

\* Sets the name of the player

\* @param xName New name for the player

\*/

public void setName(String xName) {

name = xName;

}

/\*\*

\* Sets the position of the player

\* @param xPosition New position for the player

\*/

public void setPositionPlayed(String xPosition) {

positionPlayed = xPosition;

}

/\*\*

\* Sets the number of the player.

\* Number must be an integer 1-99, else throws exception

\* @param xNumber

\*/

public void setJerseyNumber(int xNumber) {

if (xNumber < 0 || xNumber > 99)

throw new IllegalArgumentException("Number must be a "

+ "whole number 1-99");

else

jerseyNumber = xNumber;

}

/\*\*

\* Checks the contents of two Player objects for equality.<BR>

\* Returns true if objects contents are equal, false if not

\* @param xObj Object to be compared

\* @return True if objects are equal, false if not

\*/

@Override

public boolean equals(Object xObj) {

if(!(xObj instanceof Player))

return false;

Player obj = (Player)xObj;

if(!(name.equals(obj.getName())))

return false;

if(!(positionPlayed.equals(obj.getPositionPlayed())))

return false;

return jerseyNumber == obj.getJerseyNumber();

}

/\*\*

\* Returns a string representation of the data contained in the object

\* @return Single printable string of data stored in the object

\*/

@Override

public String toString() {

String output = "";

output += "Name: " + this.name;

output += "\nPosition: " + this.positionPlayed;

output += "\nJersey Number: " + this.jerseyNumber;

return output;

}

}

package lab03;

import java.util.Scanner;

/\*\*

\* Demonstrates the ArrayBag class.

\* @author Jaden Young

\*/

public class NDSUBasketBall {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

ArrayBag team = new ArrayBag();

final int NUM\_PLAYERS = 14; //magic numbers are icky

System.out.println("--------- NDSU Basketball Team ---------");

for(int i = 0; i < NUM\_PLAYERS; i++) {

System.out.println("Player " + i + ":");

// get name

System.out.print("Name > ");

String name = scan.nextLine();

// get position

System.out.print("Position > ");

String position = scan.nextLine();

// get number

System.out.print("Jersey number > ");

while(!scan.hasNextInt()) {

String garbage = scan.nextLine();

System.out.print("Jersey number > ");

}

int number = scan.nextInt();

// add player to team

team.add(new Player(name, position, number));

//eats extra character left behind

scan.nextLine();

}

//remove a random player from the team

team.remove();

//add a new player with madeup info

Player me = new Player("Jaden Young", "C", 11);

team.add(me);

System.out.println("Size of the team with all players and made up "

+ "player: " + team.getCurrentSize());

//remove the added player

team.remove(me);

System.out.println("Size of the team after removing the made up "

+ "player: " + team.getCurrentSize());

// print out all players on the team

for(int i = 0; i < team.getCurrentSize(); i++) {

try {

System.out.println(team.getItem(i).toString());

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println(e.toString());

}

}

ArrayBag courses = new ArrayBag();

courses.add("MATH 166");

courses.add("SPAN 312");

courses.add("CSCI 161");

courses.add("HNES 111");

courses.add("COMM 111");

System.out.println("Before removal of random course:");

for(int i = 0; i < courses.getCurrentSize(); i++) {

System.out.println(courses.getItem(i));

}

courses.remove();

System.out.println("After removal of random course:");

for(int i = 0; i < courses.getCurrentSize(); i++) {

System.out.println(courses.getItem(i));

}

}

}

run:

--------- NDSU Basketball Team ---------

Player 0:

Name > Brian Ishola

Position > G/F

Jersey number > 0

Player 1:

Name > Paul Miller

Position > G

Jersey number > 2

Player 2:

Name > Jake Showalter

Position > G

Jersey number > 3

Player 3:

Name > Zach checkal

Position > G

Jersey number > 4

Player 4:

Name > Lawrence Alexander

Position > G

Jersey number > 12

Player 5:

Name > Carlin Dupree

Position > G

Jersey number > 13

Player 6:

Name > A.J. Jacobson

Position > G/F

Jersey number > 21

Player 7:

Name > Kory Brown

Position > G

Jersey number > 22

Player 8:

Name > Trey Miller

Position > G/F

Jersey number > 24

Player 9:

Name > Spencer Eliason

Position > F

Jersey number > 30

Player 10:

Name > Evan Wesenberg

Position > F

Jersey number > 32

Player 11:

Name > Chris Kading

Position > F

Jersey number > 34

Player 12:

Name > Dexter Werner

Position > F

Jersey number > 40

Player 13:

Name > Matt Kourouma

Position > G/F

Jersey number > 44

Size of the team with all players and made up player: 14

Size of the team after removing the made up player: 13

Name: Brian Ishola

Position: G/F

Jersey Number: 0

Name: Paul Miller

Position: G

Jersey Number: 2

Name: Jake Showalter

Position: G

Jersey Number: 3

Name: Zach checkal

Position: G

Jersey Number: 4

Name: Lawrence Alexander

Position: G

Jersey Number: 12

Name: Carlin Dupree

Position: G

Jersey Number: 13

Name: A.J. Jacobson

Position: G/F

Jersey Number: 21

Name: Kory Brown

Position: G

Jersey Number: 22

Name: Trey Miller

Position: G/F

Jersey Number: 24

Name: Evan Wesenberg

Position: F

Jersey Number: 32

Name: Chris Kading

Position: F

Jersey Number: 34

Name: Dexter Werner

Position: F

Jersey Number: 40

Name: Matt Kourouma

Position: G/F

Jersey Number: 44

Before removal of random course:

MATH 166

SPAN 312

CSCI 161

HNES 111

COMM 111

After removal of random course:

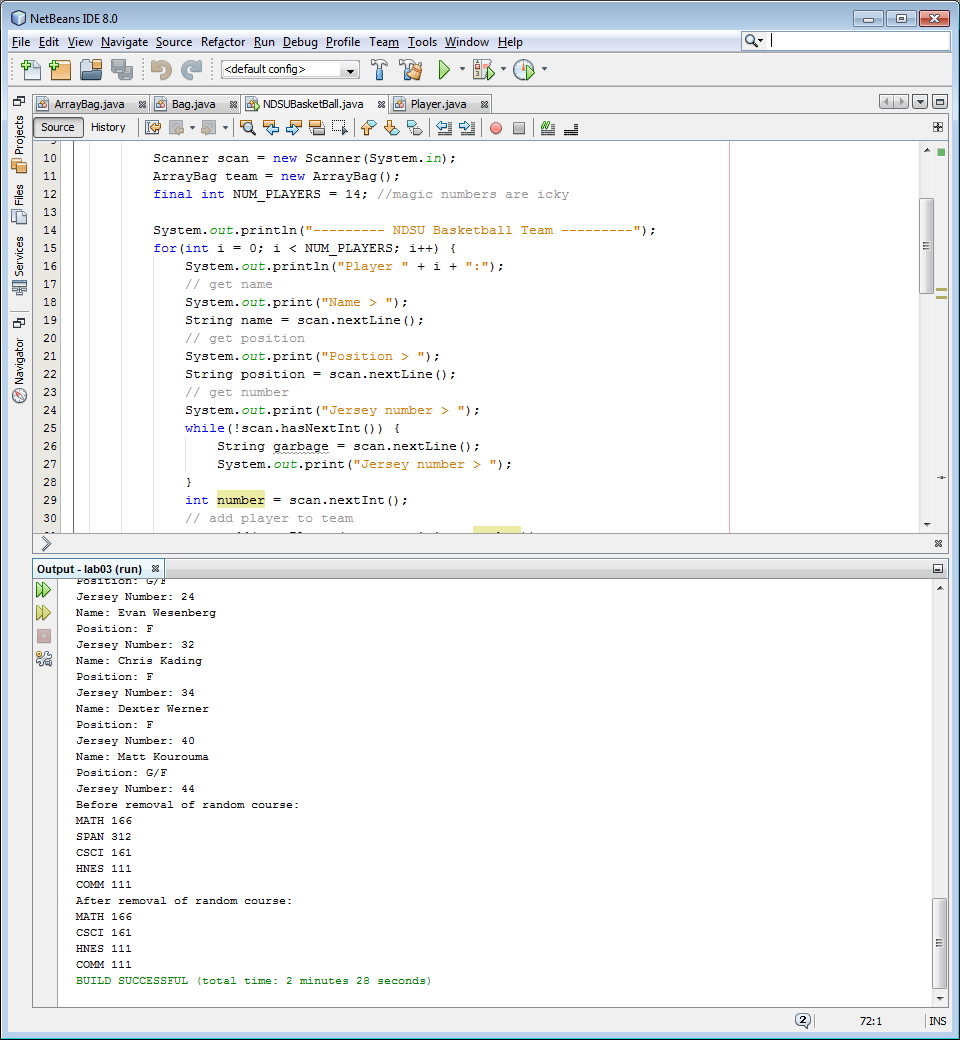
MATH 166

CSCI 161

HNES 111

COMM 111

BUILD SUCCESSFUL (total time: 2 minutes 28 seconds)





You said in class that association relationships are supposed to be represented by a dashed line, but the Visio template provides a solid line with no arrows as the association indicator. I’m going to go with that, only because I couldn’t find a dashed line.