Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Основи об'єктно-орієнтованого програмування

Лабораторна робота №6

«Робота з колекціями в мові програмування Java»

Виконала:

студентка групи ІВ-71

Молчанова В. С.

Залікова книжка № ІВ-7110

Перевірив Подрубайло О. О.

Київ

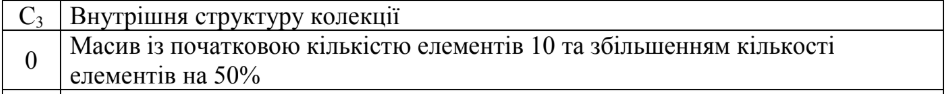
2018 р.

# Власний варіант:

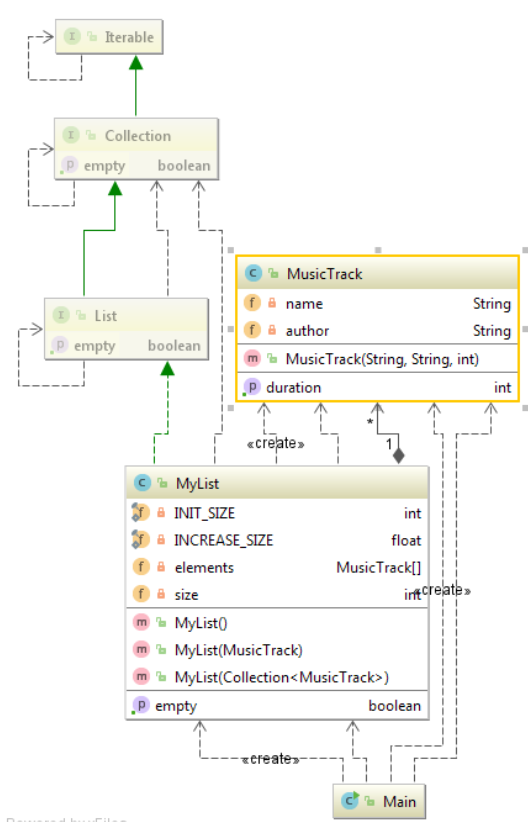
С2 = 7110 % 2 = 0

C3 = 7110 % 3 = 0





# Діаграма класів:



# Код програми:

*/\*\*  
 \* Describes a music track.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** MusicTrack **implements** Comparable<MusicTrack>{  
 **private** String **name**;  
 **private** String **author**;  
 **private int duration**;  
  
 */\*\*  
 \** ***@param name*** *name of the track  
 \** ***@param author*** *author of the track  
 \** ***@param duration*** *duration of the track in seconds  
 \*/* **public** MusicTrack(String name, String author, **int** duration){  
 **if**(duration <= 0){  
 **throw new** IllegalArgumentException(**"Track's duration must be a positive number"**);  
 }  
 **this**.**name** = name;  
 **this**.**author** = author;  
 **this**.**duration** = duration;  
 }  
  
 */\*\*  
 \** ***@return*** *author, name and style of the track  
 \*/* @Override  
 **public** String toString(){  
 String className = getClass().getName();  
 **return author** + **" - "** + **name**;  
 }  
  
 */\*\*  
 \* Compares current track and another one by style  
 \** ***@param track*** *another track  
 \*/* @Override  
 **public int** compareTo(MusicTrack track){  
 **return this**.getClass().getName().compareTo(track.getClass().getName());  
 }  
  
 */\*\*  
 \** ***@return*** *duration of the track  
 \*/* **public int** getDuration(){  
 **return duration**;  
 }  
}

**import** java.util.\*;  
  
*/\*\*  
 \* List collection implementation  
 \** ***@author*** *KindBlindMind  
 \*/***public class** MyList **implements** List<MusicTrack> {  
 **private static final int *INIT\_SIZE*** = 10;  
 **private static final float *INCREASE\_SIZE*** = 0.5f;  
 **private** MusicTrack[] **elements**;  
 **private int size**;  
  
 */\*\*  
 \* Create an empty list  
 \*/* **public** MyList() {  
 **elements** = **new** MusicTrack[***INIT\_SIZE***];  
 **size** = 0;  
 }  
  
 */\*\*  
 \* Create a list with one element  
 \** ***@param track*** *an element to add  
 \*/* **public** MyList(MusicTrack track) {  
 **this**();  
 add(0, track);  
 }  
  
 */\*\*  
 \* Create a list from a collection  
 \** ***@param collection*** *the collection  
 \*/* **public** MyList(Collection<MusicTrack> collection) {  
 **this**();  
 addAll(0, collection);  
 }  
  
 */\*\*  
 \* Get the number of elements stored in the list  
 \** ***@return*** *the size of the set  
 \*/* @Override  
 **public int** size() {  
 **return size**;  
 }  
  
 */\*\*  
 \* Checks if the set is empty  
 \** ***@return*** *True if the set is empty  
 \*/* @Override  
 **public boolean** isEmpty() {  
 **return size** == 0;  
 }  
  
 */\*\*  
 \* Checks if the set contains an item  
 \** ***@param o*** *Item to check  
 \** ***@return*** *True if the set contains the item  
 \*/* @Override  
 **public boolean** contains(Object o) {  
 **return** indexOf(o) != -1;  
 }  
  
 */\*\*  
 \* Get forward iterator for the set  
 \** ***@return*** *forward iterator object  
 \*/* @Override  
 **public** Iterator<MusicTrack> iterator() {  
 **return new** Iterator<MusicTrack>() {  
 **private int currentIndex** = 0;  
  
 @Override  
 **public boolean** hasNext() {  
 **return currentIndex** < **size**;  
 }  
  
 @Override  
 **public** MusicTrack next() {  
 **return elements**[**currentIndex**++];  
 }  
 };  
 }  
  
 */\*\*  
 \* Converts the list to an array of Object type  
 \** ***@return*** *the array representation of the items in the set  
 \*/* @Override  
 **public** Object[] toArray() {  
 **return elements**;  
 }  
  
 */\*\*  
 \* Converts the set to an array of E type, stored in a  
 \** ***@param a*** *Array large enough to store items of the set  
 \** ***@param <T>*** *Type parameter  
 \** ***@return*** *reference to the array  
 \*/* @Override  
 **public** <T> T[] toArray(T[] a) {  
 **for**(**int** i = 0; i < **size**; i++){  
 a[i] = (T)**elements**[i];  
 }  
 **return** a;  
 }  
  
 */\*\*  
 \* Adds an element to the list  
 \** ***@param track*** *track to be added  
 \** ***@return*** *True if item added  
 \*/* @Override  
 **public boolean** add(MusicTrack track) {  
 **if** (**size** == **elements**.**length**) {  
 resize();  
 }  
 **elements**[**size**++] = track;  
 **return true**;  
 }  
  
 */\*\*  
 \* Inserts the specified element at the specified position in this list  
 \** ***@param index*** *index at which the specified element is to be inserted  
 \** ***@param element*** *element to be inserted  
 \*/* @Override  
 **public void** add(**int** index, MusicTrack element) {  
 MusicTrack[] newElements = **new** MusicTrack[**elements**.**length**];  
 **for**(**int** i = 0; i < index; i++){  
 newElements[i] = **elements**[i];  
 }  
 newElements[index] = element;  
 **size**++;  
 **for**(**int** i = index + 1; i < **size**; i++){  
 **if**(i >= **size**){  
 resize();  
 }  
 newElements[i] = **elements**[i - 1];  
 }  
 **elements** = newElements;  
 }  
  
 */\*\*  
 \*Increase the size of the collection in INCREASE\_SIZE times  
 \*/* **private void** resize() {  
 **int** newSize = (**int**)(**elements**.**length** \*  
 (1+***INCREASE\_SIZE***));  
 MusicTrack[] newElements = **new** MusicTrack[newSize];  
 **for** (**int** i = 0; i < **elements**.**length**; i++) {  
 newElements[i] = **elements**[i];  
 }  
 **elements** = newElements;  
 }  
  
 */\*\*  
 \* Removes an element from the set  
 \** ***@param o*** *Element to be deleted  
 \** ***@return*** *True if the item is deleted  
 \*/* @Override  
 **public boolean** remove(Object o) {  
 **int** index = indexOf(o);  
 **if**(index == -1){  
 **return false**;  
 }  
 remove(index);  
 **return true**;  
 }  
  
 */\*\*  
 \* Checks if the set contains items from a collection  
 \** ***@param c*** *Collection to check  
 \** ***@return*** *True if the set contains the items  
 \*/* @Override  
 **public boolean** containsAll(Collection<?> c) {  
 **boolean** flag = **false**;  
 **for**(Object o : c.toArray()){  
 flag = **this**.contains(o);  
 }  
 **return** flag;  
 }  
  
 */\*\*  
 \* Add elements from the collection  
 \** ***@param c*** *Collection added  
 \** ***@return*** *True if elements added  
 \*/* @Override  
 **public boolean** addAll(Collection<? **extends** MusicTrack> c) {  
 **boolean** flag = **false**;  
 **for**(Object o : c.toArray()){  
 flag = **this**.add((MusicTrack)o);  
 }  
 **return** flag;  
 }  
  
 */\*\*  
 \* Add elements from the collection  
 \** ***@param index*** *\** ***@param c*** *\** ***@return*** *\*/* @Override  
 **public boolean** addAll(**int** index, Collection<? **extends** MusicTrack> c) {  
 **boolean** flag = **false**;  
 **for**(Object o : c){  
 *// System.out.println(elements[index]);* add(index++, (MusicTrack)o);  
 *// System.out.println(elements[index]);* }  
 **return** flag;  
 }  
  
 */\*\*  
 \* Removes elements of the collection from the list  
 \** ***@param c*** *Collection to be deleted  
 \** ***@return*** *True if any item is deleted  
 \*/* @Override  
 **public boolean** removeAll(Collection<?> c) {  
 **boolean** flag = **false**;  
 **for**(Object o : c){  
 **if**(remove(o)){  
 flag = **true**;  
 }  
 }  
 **return** flag;  
 }  
  
 */\*\*  
 \* Removes all elements of the set not present in a collection  
 \** ***@param c*** *Collection of elements to retain  
 \** ***@return*** *True if any item is deleted  
 \*/* @Override  
 **public boolean** retainAll(Collection<?> c) {  
 **boolean** flag = **false**;  
 **for**(Object o : **elements**){  
 **if**(!c.contains(o)){  
 remove(o);  
 flag = **true**;  
 }  
 }  
 **return** flag;  
 }  
  
 */\*\*  
 \* Clears the list  
 \*/* @Override  
 **public void** clear() {  
 **elements** = **new** MusicTrack[0];  
 }  
  
 */\*\*  
 \* Return element with certain index  
 \** ***@param index*** *index of element to be returned  
 \*/* @Override  
 **public** MusicTrack get(**int** index) {  
 **return elements**[index];  
 }  
  
 */\*\*  
 \* Replaces the element at the specified position in this list with the specified element  
 \** ***@param index*** *index of the element to replace  
 \** ***@param element*** *element to be stored at the specified position  
 \** ***@return*** *the element previously at the specified position  
 \*/* @Override  
 **public** MusicTrack set(**int** index, MusicTrack element) {  
 MusicTrack temp = **elements**[index];  
 **elements**[index] = element;  
 **return** temp;  
 }  
  
 */\*\*  
 \* Removes the element at the specified position in this list  
 \** ***@param index*** *the index of the element to be removed  
 \** ***@return*** *the element previously at the specified position  
 \*/* @Override  
 **public** MusicTrack remove(**int** index) {  
 MusicTrack temp = **elements**[index];  
 MusicTrack[] newElements = **new** MusicTrack[--**size**];  
 **for**(**int** i = 0; i < index; i++){  
 newElements[i] = **elements**[i];  
 }  
 **for**(**int** i = index; i < **size**; i++){  
 newElements[i] = **elements**[i+1];  
 }  
 **return** temp;  
 }  
  
 */\*\*  
 \* Returns the index of the first occurrence of the specified element in this list  
 \* or -1 if this list does not contain the element.  
 \** ***@param o*** *element to search for  
 \*/* @Override  
 **public int** indexOf(Object o) {  
 **for**(**int** i = 0; i < **size**; i++){  
 **if**(**elements**[i].equals(o)){  
 **return** i;  
 }  
 }  
 **return** -1;  
 }  
  
 */\*\*  
 \* Returns the index of the last occurrence of the specified element in this list  
 \* or -1 if this list does not contain the element.  
 \** ***@param o*** *element to search for  
 \*/* @Override  
 **public int** lastIndexOf(Object o) {  
 **for**(**int** i = **size** - 1; i >= 0; i--){  
 **if**(**elements**[i].equals(o)){  
 **return** i;  
 }  
 }  
 **return** -1;  
 }  
  
 */\*\*  
 \* Returns a list iterator over the elements in this list  
 \*/* @Override  
 **public** ListIterator<MusicTrack> listIterator() {  
 **return new** ListIterator<MusicTrack>() {  
 **private int currentIndex** = 0;  
  
 @Override  
 **public boolean** hasNext() {  
 **return currentIndex** < **size** - 1;  
 }  
  
 @Override  
 **public** MusicTrack next() {  
 **return elements**[**currentIndex**++];  
 }  
  
 @Override  
 **public boolean** hasPrevious() {  
 **return currentIndex** > 0;  
 }  
  
 @Override  
 **public** MusicTrack previous() {  
 **return elements**[**currentIndex**--];  
 }  
  
 @Override  
 **public int** nextIndex() {  
 **return** (hasNext() ? **currentIndex** + 1 : **size**);  
 }  
  
 @Override  
 **public int** previousIndex() {  
 **return** (hasPrevious() ? **currentIndex** - 1 : -1);  
 }  
  
 @Override  
 **public void** remove() {  
 MyList.**this**.remove(**currentIndex**);  
 }  
  
 @Override  
 **public void** set(MusicTrack musicTrack) {  
 MyList.**this**.set(**currentIndex**, musicTrack);  
 }  
  
 @Override  
 **public void** add(MusicTrack musicTrack) {  
 MyList.**this**.add(nextIndex(), musicTrack);  
 }  
 };  
 }  
  
 */\*\*  
 \* Returns a list iterator over the elements in this list, starting at the specified position in the list  
 \** ***@param index*** *index of the first element to be returned from the list iterator  
 \*/* @Override  
 **public** ListIterator<MusicTrack> listIterator(**int** index) {  
 **return new** ListIterator<MusicTrack>() {  
 **private int currentIndex** = index;  
  
 @Override  
 **public boolean** hasNext() {  
 **return currentIndex** < **size** - 1;  
 }  
  
 @Override  
 **public** MusicTrack next() {  
 **return elements**[**currentIndex**++];  
 }  
  
 @Override  
 **public boolean** hasPrevious() {  
 **return currentIndex** > 0;  
 }  
  
 @Override  
 **public** MusicTrack previous() {  
 **return elements**[**currentIndex**--];  
 }  
  
 @Override  
 **public int** nextIndex() {  
 **return** (hasNext() ? **currentIndex** + 1 : **size**);  
 }  
  
 @Override  
 **public int** previousIndex() {  
 **return** (hasPrevious() ? **currentIndex** - 1 : -1);  
 }  
  
 @Override  
 **public void** remove() {  
 MyList.**this**.remove(**currentIndex**);  
 }  
  
 @Override  
 **public void** set(MusicTrack musicTrack) {  
 MyList.**this**.set(**currentIndex**, musicTrack);  
 }  
  
 @Override  
 **public void** add(MusicTrack musicTrack) {  
 MyList.**this**.add(nextIndex(), musicTrack);  
 }  
 };  
 }  
  
 */\*\*  
 \* Returns a view of the portion of this list between the specified fromIndex and toIndex  
 \** ***@param fromIndex*** *low endpoint (inclusive) of the subList  
 \** ***@param toIndex*** *high endpoint (exclusive) of the subLis  
 \*/* @Override  
 **public** MyList subList(**int** fromIndex, **int** toIndex) {  
 MusicTrack[] newElements = **new** MusicTrack[toIndex - fromIndex];  
 **for**(**int** i = 0; i < toIndex - fromIndex; i++){  
 newElements[i] = **elements**[fromIndex + i];  
 };  
 **return new** MyList(Arrays.*asList*(newElements));  
 }  
  
  
}

*/\*\*  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***import** java.util.Arrays;  
**import** java.util.List;  
**public class** Main {  
 **public static void** main(String[] args) {  
 MusicTrack song1 = **new** MusicTrack(**"Zombie"**, **"The Cranberries"**, 315);  
 MusicTrack song2 = **new** MusicTrack(**"Bohemian Rhapsody"**, **"Queen"**, 367);  
 MusicTrack song3 = **new** MusicTrack(**"Dancing Queen"**, **"ABBA"**, 232);  
 MusicTrack song4 = **new** MusicTrack(**"Winter"**, **"Vivaldi"**, 570);  
  
 MyList myList = **new** MyList(song1);  
 myList.add(song2);  
 MusicTrack[] array = {song3, song4};  
 myList.addAll(1,Arrays.*asList*(array));  
 **for**(MusicTrack track : myList){  
 System.***out***.println(track);  
 }  
 }  
}