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 №5

«Наслідування та поліморфізм»

Виконала:

студентка групи ІВ-71

Молчанова В. С.

Залікова книжка № ІВ-7110

Перевірив Подрубайло О. О.

Київ

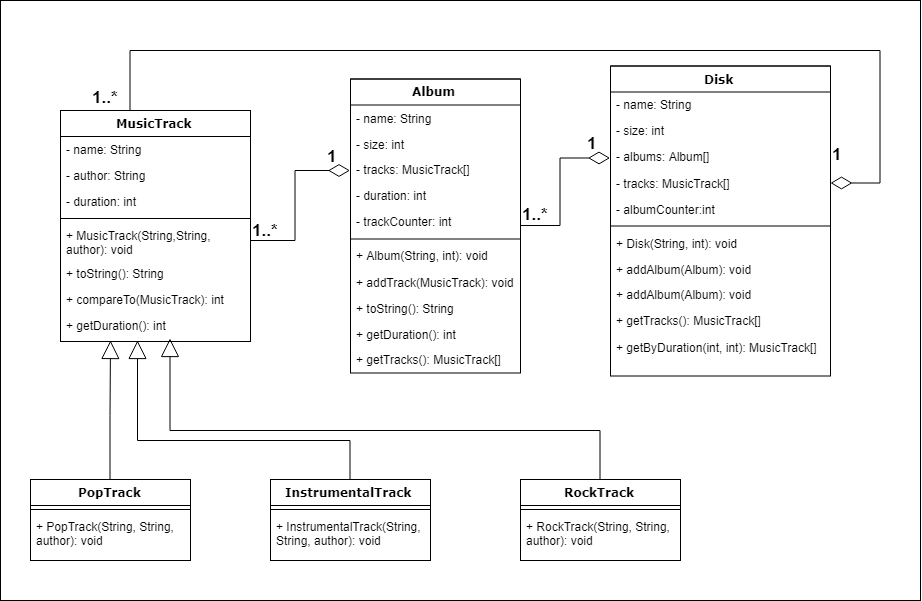
2018 р.

# Власний варіант:

С17 = 7110 % 17 = 4

|  |  |
| --- | --- |
| 4 | Визначити ієрархію музичних композицій. Записати на диск альбом.  Порахувати тривалість альбому. Провести перестановку композицій диска на  основі приналежності до стилю. Знайти композицію, що відповідає заданому  діапазону довжини треків. |

# Діаграма класів:



# Код програми:

*/\*\*  
 \* This program allows to create albums with songs of different styles, write them to disks,  
 \* sort tracks by the style and find them by duration.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** Main {  
  
 **public static void** main(String[] args) {  
 RockTrack song1 = **new** RockTrack(**"Zombie"**, **"The Cranberries"**, 315);  
 RockTrack song2 = **new** RockTrack(**"Bohemian Rhapsody"**, **"Queen"**, 367);  
 PopTrack song3 = **new** PopTrack(**"Dancing Queen"**, **"ABBA"**, 232);  
 InstrumentalTrack song4 = **new** InstrumentalTrack(**"Winter"**, **"Vivaldi"**, 570);  
  
 Album album = **new** Album(**"album"**, 4);  
 album.addTrack(song1);  
 album.addTrack(song3);  
 album.addTrack(song2);  
 album.addTrack(song4);  
  
 Disk disk = **new** Disk(**"disk"**, 1);  
 disk.addAlbum(album);  
 System.***out***.println(**"\* Songs on the disk:"**);  
 **for**(MusicTrack i: disk.getTracks()){  
 System.***out***.println(i);  
 }  
 System.***out***.println(**"\n\* Duration of the album:"**);  
 System.***out***.println(album.getDuration() + **" seconds"**);  
  
 disk.sortByStyle();  
 System.***out***.println(**"\n\* Songs on the disk, sorted by style:"**);  
 **for**(MusicTrack i: disk.getTracks()){  
 System.***out***.println(i);  
 }  
  
 System.***out***.println(**"\n\* Songs on the disk with duration from 300 to 400 seconds:"**);  
 MusicTrack[] from300to400 = disk.getByDuration(300, 400);  
 **if** (from300to400.**length** == 0){  
 System.***out***.println(**"No such songs"**);  
 }  
 **else**{  
 **for**(MusicTrack i: from300to400){  
 System.***out***.println(i);  
 }  
 }  
 }  
}

**import** java.util.Arrays;  
  
*/\*\*  
 \* Describes a disk with possibility to write some albums on it.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** Disk {  
 **private** String **name**;  
 **private int size**;  
 **private** Album[] **albums**;  
 **private** MusicTrack[] **tracks** = {};  
 **private int albumCounter** = 0;  
  
 */\*\*  
 \** ***@param name*** *name of the disk  
 \** ***@param size*** *the number of albums  
 \*/* **public** Disk(String name, **int** size){  
 **if**(size <= 0){  
 **throw new** IllegalArgumentException(**"Disk's size must be a positive number"**);  
 }  
 **this**.**name** = name;  
 **this**.**size** = size;  
 **albums** = **new** Album[size];  
 }  
  
 */\*\*  
 \* Writes album on the disk  
 \** ***@param album*** *the album to be written  
 \*/* **public void** addAlbum(Album album){  
 **if**(**albumCounter** >= **size**){  
 System.***out***.println(**"! Album not added (disk is full)"**);  
 }  
 **else**{  
 **albums**[**albumCounter**++] = album;  
 **for**(MusicTrack track: album.getTracks()){  
 **tracks** = Arrays.*copyOf*(**tracks**, **tracks**.**length** + 1);  
 **tracks**[**tracks**.**length** - 1] = track;  
 }  
 }  
 }  
  
 */\*\*  
 \* Sorts tracks on disk by style  
 \*/* **public void** sortByStyle(){  
 **for**(**int** i = 0; i < **tracks**.**length** - 1; i++){  
 **for**(**int** j = i + 1; j < **tracks**.**length**; j++){  
 **if**(**tracks**[i].compareTo(**tracks**[j]) > 0){  
 MusicTrack temp = **tracks**[i];  
 **tracks**[i] = **tracks**[j];  
 **tracks**[j] = temp;  
 }  
 }  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *tracks from the disk  
 \*/* **public** MusicTrack[] getTracks() {  
 **return tracks**;  
 }  
  
 */\*\*  
 \** ***@param min*** *minimal duration  
 \** ***@param max*** *maximal duration  
 \** ***@return*** *array of MusicTracks with fee from minFee to maxFee  
 \*/* **public** MusicTrack[] getByDuration(**int** min, **int** max) {  
 MusicTrack[] result = {};  
  
 **for** (MusicTrack MusicTrack : **tracks**) {  
 **if** (MusicTrack.getDuration() >= min && MusicTrack.getDuration() <= max) {  
 result = Arrays.*copyOf*(result, result.**length** + 1);  
 result[result.**length** - 1] = MusicTrack;  
 }  
 }  
  
 **return** result;  
 }  
}

*/\*\*  
 \* Describes an album with possibility to add some music tracks to it.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** Album {  
 **private** String **name**;  
 **private int size**;  
 **private** MusicTrack[] **tracks**;  
 **private int duration** = 0;  
 **private int trackCounter**;  
  
 */\*\*  
 \** ***@param name*** *name of the album  
 \** ***@param size*** *the number of songs in the album  
 \*/* **public** Album(String name, **int** size){  
 **if**(size <= 0){  
 **throw new** IllegalArgumentException(**"Album's size must be a positive number"**);  
 }  
 **this**.**name** = name;  
 **this**.**size** = size;  
 **tracks** = **new** MusicTrack[size];  
 **trackCounter** = 0;  
 }  
  
 */\*\*  
 \* Adds a music track to the album  
 \** ***@param track*** *a track to be added  
 \*/* **public void** addTrack(MusicTrack track){  
 **if**(**trackCounter** < **size**){  
 **tracks**[**trackCounter**++] = track;  
 **duration** += track.getDuration();  
 }  
 **else**{  
 System.***out***.println(**"! Song not added (album is full)"**);  
 }  
 }  
  
 */\*\*  
 \** ***@return*** *name of the album and number of tracks in it.  
 \*/* @Override  
 **public** String toString(){  
 **return name** + **" ("** + **trackCounter** + **" tracks)"**;  
 }  
  
 */\*\*  
 \** ***@return*** *duration of the album  
 \*/* **public int** getDuration(){  
 **return duration**;  
 }  
  
 */\*\*  
 \** ***@return*** *tracks from the album  
 \*/* **public** MusicTrack[] getTracks() {  
 **return tracks**;  
 }  
}

*/\*\*  
 \* Describes an abstract music track.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public abstract 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** + **" ("** + className.substring(0, className.length() - 5) + **")"**;  
 }  
  
 */\*\*  
 \* 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**;  
 }  
}

*/\*\*  
 \* Extends abstract music track and describes instrumental track.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** InstrumentalTrack **extends** MusicTrack {  
 **public** InstrumentalTrack(String name, String author, **int** duration){  
 **super**(name, author, duration);  
 }  
}

*/\*\*  
 \* Extends abstract music track and describes pop track.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** PopTrack **extends** MusicTrack {  
 **public** PopTrack(String name, String author, **int** duration){  
 **super**(name, author, duration);  
 }  
}

*/\*\*  
 \* Extends abstract music track and describes rock track.  
 \** ***@author*** *KindBlindMind  
 \** ***@version*** *1.0  
 \*/***public class** RockTrack **extends** MusicTrack {  
 **public** RockTrack(String name, String author, **int** duration){  
 **super**(name, author, duration);  
 }  
}