**ArrayList java apl**

**Api 사용해보기 (1)**

ArrayList는 java.util.ArrayList에 포함되어 있습니다. ArrayList를 사용하기 위해 java.util.ArrayList를 import하세요.

해답설명

|  |  |
| --- | --- |
| 3  4  5  6  7  8  9  10 | // ArrayList를 import 하세요.  import  public class MainRunner  {  public static void main(String[] args)  {  return ;  }  } |

**Api 사용해보기 (2)**

이제 ArrayList 객체를 만들어 봅시다. 빈칸을 채워 String을 담는 ArrayList인 myarrlist를 만들어보세요.

예를 들어, Integer를 담는 ArrayList인 numbers를 만드는 방법은 아래와 같습니다.

ArrayList<Integer> numbers = new ArrayList<>();

해답설명

|  |
| --- |
|  |

import java.util.ArrayList;

public class MainRunner

{

public static void main(String[] args)

{

ArrayList<  > my\_arr\_list = new 

return;

}

}

**Api 사용해보기 (3)**

myarrlist에 "hello", "java", "world!" 를 추가해봅시다. add 메소드를 쓰면 ArrayList에 원소를 추가할 수 있습니다.

엘리먼트를 추가할 때는 add 메소드를 사용합니다. 아래의 예를 참고하여 문제를 해결해 보세요.

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

해답설명

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15 | import java.util.ArrayList;  public class MainRunner  {  public static void main(String[] args)  {  ArrayList<String> my\_arr\_list = new ArrayList<String>();  // add 메소드를 세 번 호출해 "hello", "java", "world!"를 추가해보세요.        } |

**Api 사용해보기 (4)**

특정 인덱스에 위치하는 엘리먼트를 삭제할 때는 remove를 사용합니다. remove를 사용해 2번째 원소를 삭제하세요.

remove는 아래와 같이 사용합니다. 아래의 예를 참고하여 문제를 해결해 보세요.

numbers.remove(3); // numbers의 세 번째 원소를 삭제합니다.

해답 설명

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import java.util.ArrayList;  public class MainRunner  {  public static void main(String[] args)  {  ArrayList<String> my\_arr\_list = new ArrayList<String>();  my\_arr\_list.add("hello");  my\_arr\_list.add("java");  my\_arr\_list.add("world!");  // 2번째 원소인 "world!"를 삭제하세요.    }  } |

**Api 사용해보기 (5)**

엘리먼트를 가져올 때는 get을 사용합니다. get을 써 myarrlist의 1번째 원소를 출력해보세요.

특정 인덱스에 위치한 엘리먼트를 가져오기 위해 get을 이용하는 문제입니다. 아래의 예를 참고하여 문제를 해결해 보세요.

numbers.get(2); // numbers의 두 번째 엘리먼트를 가져옵니다.

해답설명

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | import java.util.ArrayList;  public class MainRunner  {  public static void main(String[] args)  {  ArrayList<String> my\_arr\_list = new ArrayList<String>();  my\_arr\_list.add("hello");  my\_arr\_list.add("java");  my\_arr\_list.add("world!");  // my\_arr\_list의 1번째 원소 "java"를 출력해보세요.  System.out.println(  );  }  } |

**자바 api 반복문 쓰기 (1)**

자바에서는 ArrayList를 탐색하기 위한 방법으로 iterator을 제공합니다. 이것은 주로 객체지향 프로그래밍에서 사용하는 반복기법입니다. myarraylist의 Iterator 객체 it을 만들어 주세요.

해답설명

import java.util.ArrayList;

import java.util.Iterator;

public class MainRunner

{

public static void main(String[] args)

{

ArrayList<String> my\_arr\_list = new ArrayList<String>();

my\_arr\_list.add("hello");

my\_arr\_list.add("java");

my\_arr\_list.add("world!");

// my\_arr\_list의 iterator it을 초기화하세요.

Iterator<  > it = 

}

}

**JAVA API 반복문쓰기 (2)**

Iterator 객체를 만들었으니, while문으로 myarrlist의 모든 원소를 출력해봅시다. 다음 가이드를 보고 코드의 빈 부분을 완성하세요.

* 이터레이터의 hasNext() 메소드를 쓰면 더 순회할 엘레먼트가 있는지 알 수 있습니다.
* 이터레이터의 it.next() 메소드를 쓰면 다음 엘레먼트를 갖고 올 수 있습니다.

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | import java.util.ArrayList;  import java.util.Iterator;  public class MainRunner  {  public static void main(String[] args)  {  ArrayList<String> my\_arr\_list = new ArrayList<String>();  my\_arr\_list.add("hello");  my\_arr\_list.add("java");  my\_arr\_list.add("world!");  Iterator<String> it = my\_arr\_list.iterator();  // while문으로 my\_arr\_list의 모든 원소를 출력해봅시다.  while(  )  {  System.out.println(  );  }  }  } |

해답설명

**ArrayList 직접 구현하기- 생성과 원소 추가**

**객체생성하기**

###### 문제 설명

이번 실습에는 배열으로 ArrayList를 구현할겁니다.

ArrayList class의 private 멤버 변수로, Object를 담는 배열 elementData를 만드세요. elementData를의 length는 자유롭게 정해보세요.

해답 설명

|  |  |
| --- | --- |
| 2  3  4  5  6 | class ArrayList  {  private int size = 0;  // object[] 타입 private 멤버변수 elementData를 만드세요  private  elementData =  } |

**데이터 추가하기- 마지막위치에 추가하기(1)**

###### 문제 설명

Object 타입 원소 하나를 맨 마지막 위치에 추가하는 메소드 addLast를 만들려고합니다.

우선, Object 타입을 인자로 받는 빈 public 메소드, addLast를 만들어 보세요.

해답 설명

public class MainRunner

{

public static void main(String[] args)

{

ArrayList arraylist = new ArrayList();

arraylist.addLast(30);

return ;

}

}

**마지막 위치에 추가하기(2)**

###### 문제 설명

Object 타입 원소 하나를 맨 마지막 위치에 추가하는 메소드 addLast를 만들려고합니다. 빈칸을 알맞게 채워 문제를 해결해 보세요

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

public boolean addLast(Object e)

{

// elementData의 마지막 위치에 인자 e를 추가하세요.

elementData[  ] = 

// size를 1 증가시키세요.



return true;

}

}

**데이터추가하기 – 중간위치에 추가하기(1)**

###### 문제 설명

데이터를 중간에 추가하려면 데이터를 추가하려는 위치와 추가하려는 데이터가 필요합니다.  
데이터를 추가하려는 위치인 index(int 타입)과 추가하려는 데이터 element(Object 타입)를 인자로 받는 add 함수를 완성해보세요. add 함수는 public함수이며, boolean을 리턴합니다.

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

public boolean addLast(Object element)

{

elementData[size++] = element;

return true;

}

// int 타입 index와 Object 타입 element를 인자로 받는 public 함수 add를 완성하세요.

// add 함수는 boolean을 리턴합니다.

public 

{

return true;

}

}

**중간위치에 추가하기(2)**

###### 문제 설명

데이터를 중간에 추가하려면 먼저 데이터를 추가할 빈 공간을 확보해야 합니다. index번째 위치에 빈 공간을 확보하려면, 원래 index번째에 있던 데이터와, 그 뒤에 있던 데이터를 뒤로 밀어줘야합니다.

for문을 이용해, 끝부터 index번째까지의 데이터를 한칸씩 뒤로 이동시키는 코드를 작성해보세요.

해답설명

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class ArrayList {  private int size = 0;  private Object[] elementData = new Object[50];  public boolean addLast(Object e)  {  elementData[size++] = e;  return true;  }  public boolean add(int index, Object element)  {  //끝부터 index번째까지의 데이터를 한칸씩 뒤로 이동시키세요.  for(int i=size-1;  ; i--)  {  elementData[  ] = elementData[  ];  }  return true;  }  } |

**중간위치에 추가하기 (3)**

###### 문제 설명

이제 빈 공간을 확보했으니, index에 노드를 추가해봅시다. addLast때와 마찬가지로, 마지막에 size를 하나 증가시키는 것도 잊지 마세요.

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

public boolean addLast(Object e)

{

elementData[size++] = e;

return true;

}

public boolean add(int index, Object element)

{

for (int i = size - 1; i >= index; i--) {

elementData[i + 1] = elementData[i];

}

//elementData의 index에 데이터를 추가하세요.



//size를 1 증가하세요.



return true;

}

}

**데이터 추가하기 – 처음에 추가하기**

###### 문제 설명

이번엔 첫번째 위치에 데이터를 추가하는 addFirst 메소드를 구현해 봅시다. addFirst 메소드는 모든 데이터를 뒤로 한 칸씩 민 후, 새로운 데이터를 맨 앞에 추가해야하는데요. add 함수와 왠지 비슷하지 않나요? add 메소드를 이용해 addFirst 함수를 구현해보세요.

해답설명

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | class ArrayList {  private int size = 0;  private Object[] elementData = new Object[50];  public boolean addLast(Object e)  {  elementData[size++] = e;  return true;  }  public boolean add(int index, Object element)  {  for (int i = size - 1; i >= index; i--) {  elementData[i + 1] = elementData[i];  }  elementData[index] = element;  size++;  return true;  }  public boolean addFirst(Object element)  {  // add 메소드를 이용해서 데이터를 첫번째 위치에 저장하는 addFirst함수를 구현하세요.  return add(  );  }  } |

**ArrayList 직접 구현하기- 데이터 확인하기**

**데이터 가져오기**

###### 문제 설명

데이터를 아무리 많이 저장해도 저장한 데이터를 불러올 수 없다면 무슨 소용일까요? n번째에 위치한 데이터를 불러오는 get 메소드를 만들어 봅시다. 다음 조건에 맞춰, get 메소드 안의 빈 칸을 채워보세요.

* get 메소드는 elementData의 원소와 같은 타입을 리턴한다.
* get 메소드는 인자 index번째의 원소를 리턴한다.

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

// index에 위치한 데이터를 가져오는 get 함수를 완성하세요.

public  get(int index)

{

return  ;

}

public boolean addLast(Object e)

{

elementData[size++] = e;

return true;

}

public boolean add(int index, Object element)

{

for (int i = size - 1; i >= index; i--) {

elementData[i + 1] = elementData[i];

}

elementData[index] = element;

size++;

return true;

}

public boolean addFirst(Object element)

{

return add(0, element);

}

}

**데이터 확인하기**

###### 문제 설명

toString 메소드를 오버라이드하면 클래스 또는 객체를 대표하는 문자열을 커스터마이즈할 수 있습니다. toString 메소드를 오버라이드 해 System.out.println(arraylist)와 같이 저장된 데이터를 한 번에 확인할 수 있게 해주세요.

예를 들어 순서대로 1, 2, 3, 4, 5가 저장된 ArrayList의 경우 "[1,2,3,4,5]" 를 리턴하면 됩니다.

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

public Object get(int index)

{

return elementData[index];

}

public String toString()

{

// 저장된 데이터를 한 번에 확인할 수 있게 toString을 구현해보세요.

// 문자열을 [, , , ,] 이런 형태로 출력학기위해 스트링 선언

//그안의 배열을 폴문으로 돌려 데이터를 담는다

String str="[";

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

str += elementData[i];

if(i<(size-1) ){

str += ",";

}

}

return str+"]";

}

public boolean addLast(Object e)

{

elementData[size++] = e;

return true;

}

public boolean add(int index, Object element)

{

for (int i = size - 1; i >= index; i--) {

elementData[i + 1] = elementData[i];

}

elementData[index] = element;

size++;

return true;

}

public boolean addFirst(Object element)

{

return add(0, element);

}

}

**엘레멘트 개수 알아내기**

###### 문제 설명

엘리먼트의 개수를 int타입으로 리턴하는 퍼블릭 메소드, size를 구현해봅시다. size 메소드는 멤버 변수 size를 돌려주기만 하면 됩니다.

해답설명

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | class ArrayList {  private int size = 0;  private Object[] elementData = new Object[50];  // 빈칸을 채워 size함수를 구현하세요.  public  size()  {  // 멤버 변수 size를 리턴하세요.  return  ;  }  public Object get(int index)  {  return elementData[index];  }  public String toString()  {  String ans = "[";  for(int i=0; i<size; i++)  {  ans += elementData[i];  if(i<size-1)  {  ans += ",";  }  }  ans += "]";  return ans;  }  public boolean addLast(Object e)  {  elementData[size++] = e;  return true;  }  public boolean add(int index, Object element)  {  for (int i = size - 1; i >= index; i--) {  elementData[i + 1] = elementData[i];  }  elementData[index] = element;  size++;  return true;  }  public boolean addFirst(Object element)  {  return add(0, element);  }  } |

**ArrayList 탐색하기**

###### 문제 설명

특정한 값을 가진 엘리먼트의 인덱스값을 알아내는 indexOf 메소드를 만들어봅시다. indexOf 메소드는 다음 사항을 만족해야 합니다.

* indexOf 메소드는 int를 리턴합니다.
* indexOf 메소드는 Object 타입을 인자로 받습니다.
* indedOf 메소드는 찾는 값이 있다면 그 값이 발견되는 첫 번째 인덱스 값을 리턴하고 찾는 값이 없다면 -1을 리턴 합니다.

예를 들어 ArrayList에 [10,20,30,40,50]이 순서대로 저장되어 있다면, indexOf(20)은 1을 리턴하며, indexOf(100)은 -1을 리턴해야 합니다.

※ Object형을 비교할 때에는 비교 연산자 ==이 아니라 a.equals(b)와 같이 equals 메소드를 이용해야 합니다. 자세한 건 [자바에서 String 클래스의 equals() 메소드와 동등비교연산자 (==)의 차이가 무엇인가요?](http://hashcode.co.kr/questions/1989)를 보세요.

해답설명

class ArrayList {

private int size = 0;

private Object[] elementData = new Object[50];

public int size()

{

return size;

}

// 3가지 사항을 만족하는 indexOf 메소드를 완성하세요.

//엘리먼트의 index 검색

public int indexOf(Object o) {

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

if(o.equals(elementData[i])){

return i;

}

}

return -1; //-1인 이유는 찾는 값이 없다는 뜻

}

public Object get(int index)

{

return elementData[index];

}

public String toString()

{

String ans = "[";

for(int i=0; i<size; i++)

{

ans += elementData[i];

if(i<size-1)

{

ans += ",";

}

}

ans += "]";

return ans;

}

public boolean addLast(Object e)

{

elementData[size++] = e;

return true;

}

public boolean add(int index, Object element)

{

for (int i = size - 1; i >= index; i--) {

elementData[i + 1] = elementData[i];

}

elementData[index] = element;

size++;

return true;

}

public boolean addFirst(Object element)

{

return add(0, element);

}

}

**파트5. 과제**

**원소를 무한히 저장하는 ArrayList**

앞서 만든 ArrayList는 배열이 꽉 차면 index에러가 발생합니다. 배열이 꽉 차면 배열 크기를 늘여, 원소를 무한정 저장할 수 있게 만들어보세요.

해답설명

public class ArrayList {

private int size = 0;

private Object[] elementData = new Object[100];

public ArrayList() {

}

public boolean addLast(Object element) {

elementData[size] = element;

size++;

return true;

}

public boolean add(int index, Object element) {

// 엘리먼트 중간에 데이터를 추가하기 위해서는 끝의 엘리먼트부터 index의 노드까지 뒤로 한칸씩 이동시켜야 합니다.

for (int i = size - 1; i >= index; i--) {

elementData[i + 1] = elementData[i];

}

// index에 노드를 추가합니다.

elementData[index] = element;

// 엘리먼트의 숫자를 1 증가 시킵니다.

size++;

return true;

}

public boolean addFirst(Object element){

return add(0, element);

}

public String toString() {

String str = "[";

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

str += elementData[i];

if (i < size - 1)

str += ",";

}

return str + "]";

}

public Object remove(int index) {

// 엘리먼트를 삭제하기 전에 삭제할 데이터를 removed 변수에 저장합니다.

Object removed = elementData[index];

// 삭제된 엘리먼트 다음 엘리먼트부터 마지막 엘리먼트까지 순차적으로 이동해서 빈자리를 채웁니다.

for (int i = index + 1; i <= size - 1; i++) {

elementData[i - 1] = elementData[i];

}

// 크기를 줄입니다.

size--;

// 마지막 위치의 엘리먼트를 명시적으로 삭제해줍니다.

elementData[size] = null;

return removed;

}

public Object removeFirst(){

return remove(0);

}

public Object removeLast(){

return remove(size-1);

}

public Object get(int index) {

return elementData[index];

}

public int size() {

return size;

}

public int indexOf(Object o) {

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

if (o.equals(elementData[i])) {

return i;

}

}

return -1;

}

public ListIterator listIterator() {

// ListIterator 인스턴스를 생성해서 리턴합니다.

return new ListIterator();

}

class ListIterator {

// 현재 탐색하고 있는 순서를 가르키는 인덱스 값

private int nextIndex = 0;

// next 메소르를 호출할 수 있는지를 체크합니다.

public boolean hasNext() {

// nextIndex가 엘리먼트의 숫자보다 적다면 next를 이용해서 탐색할 엘리먼트가 존재하는 것이기 때문에 true를 리턴합니다.

return nextIndex < size();

}

// 순차적으로 엘리먼트를 탐색해서 리턴합니다.

public Object next() {

// nextIndex에 해당하는 엘리먼트를 리턴하고 nextIndex의 값을 1 증가 시킵니다.

return elementData[nextIndex++];

}

// previous 메소드를 호출해도 되는지를 체크합니다.

public boolean hasPrevious(){

// nextIndex가 0보다 크다면 이전 엘리먼트가 존재한다는 의미입니다.

return nextIndex > 0;

// 순차적으로 이전 노드를 리턴합니다.

public Object previous(){

// 이전 엘리먼트를 리턴하고 nextIndex의 값을 1감소합니다.

return elementData[--nextIndex];

}

// 현재 엘리먼트를 추가합니다.

public void add(Object element){

ArrayList.this.add(nextIndex++, element);

}

// 현재 엘리먼트를 삭제합니다.

public void remove(){

ArrayList.this.remove(nextIndex-1);

nextIndex--;

}

}

}