**Arrays 클래스**

2017-09-13

이승진

**학습목표**

가변 파라미터 메소드 문법

Arrays 클래스 메소드 목록

Arrays 클래스를 활용하여 배열의 정렬 구현

Arrays 클래스를 활용하여 배열의 이진 탐색 구현

Comparator 인터페이스 활용

**목차**

[1. 가변 파라미터 메소드 2](#_Toc493592976)

[1) 가변 파라미터 메소드란? 2](#_Toc493592977)

[2) 예제 코드 3](#_Toc493592978)

[2. 유틸러티 클래스, 헬퍼 클래스 6](#_Toc493592979)

[1) 자식 클래스 6](#_Toc493592980)

[2) 유틸러티 클래스 7](#_Toc493592981)

[3. Arrays 클래스 8](#_Toc493592982)

[1) Arrays 클래스 8](#_Toc493592983)

[2) Arrays 클래스의 메소드 9](#_Toc493592984)

[3) 예제 #1 10](#_Toc493592985)

[4) 예제 #2 11](#_Toc493592986)

[5) 예제 #3 13](#_Toc493592987)

[6) 예제 #4 16](#_Toc493592988)

[4. 과제 17](#_Toc493592989)

# 가변 파라미터 메소드

## 가변 파라미터 메소드란?

C언어의 printf 함수와 Java 언어의 System.out.printf 메소드의 파라미터 수는 고정되어 있지 않다.

이렇게 파라미터 수가 고정되어 있지 않은 메소드를 가변 파라미터 메소드라고 부른다.

혹은 가변 인자 메소드라고 부른다.

가변 파라미터 메소드의 규칙은 다음과 같다.

\* 가변 파라미터를 전달할 때, 파라미터를 0 개, 1 개, 2 개... 여러 개 전달 가능하다.

\* 가변 파라미터 변수의 실제 타입은 배열이다. 그 배열에 여러 개의 파라미터를 담아서 전달한다.

\* 그렇기 때문에, 가변 파라미터를 전달할 때, 배열 1개를 전달해도 된다.

\* 가변 파라미터는 그 메소드의 마지막 파라미터이어야 한다.

즉 고정 파라미터가 가변 파라미터 보다 앞에 있어야 한다.

가변 파라미터 전달 #1

void print1(int... param1);

print1(1); // param1 변수에 new int[] {1} 배열이 전달됨

print1(1, 2, 3); // param1 변수에 new int[] {1, 2, 3} 배열이 전달됨

print1(); // param1 변수에 빈 배열이 전달됨. new int[0]

가변 파라미터 전달 #2

void print1(int param1, int... param2);

print1(1); // param1 변수에 1 이 전달되고, param2 변수에 빈 배열이 전달됨

print1(1, 2, 3); // param1 변수에 1 이 전달되고, param2 변수에 new int[] {2, 3} 배열이 전달됨

print1(); // 컴파일 에러. 고정 파라미터인 param1은 생략할 수 없기 때문.

가변 파라미터는 메소드의 마지막 파라미터이어야 한다.

예:

void print1(String param1, String... param2); // param2가 마지막 파라미터이므로 ok

void print2(String... param1); // param1이 마지막 파라미터이므로 ok

void print3(String... param1, String param2); // param1이 마지막 파라미터가 아니므로 컴파일 에러

void print4(int param1, String... param2); // param2가 마지막 파라미터이므로 ok

void print5(int... param1); // param1이 마지막 파라미터이므로 ok

## 예제 코드

### VarArgs1.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | package net.skhu.lecture04;  public class VarArgs1 {    static void print(int... a) {  System.out.printf("\nlength=%d : ", a.length);  for (int i : a)  System.out.printf("%d ", i);  }    public static void main(String[] args) {  print(11, 2, 31, 44);  print(51, 35);  print(64);  print();  int[] a = new int[] { 11, 2, 31, 44 };  print(a);  }  } |

(줄5) void print(int... a)

print 메소드의 파라미터 a는 가변 파라미터이다.

파라미터 변수 a의 실제 타입은 int[] 이다.

print 메소드를 호출 할 때 파라미터를 0 개, 1 개, 2 개, 3 개... 여러 개로 호출해도 된다.

(줄17~18)

print 메소드의 파라미터는 가변 파라미터 변수 a 이고,

a 의 실제 타입이 배열이기 때문에,

print 메소드를 호출할 때, 파라미터로 배열을 전달해도 된다.

실행 결과 출력

|  |
| --- |
| length=4 : 11 2 31 44  length=2 : 51 35  length=1 : 64  length=0 :  length=4 : 11 2 31 44 |

### VarArgs2.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.lecture04;  public class VarArgs2 {    static void print(String a, int... b) {  System.out.printf("\na=%s, b.length=%d : ", a, b.length);  for (int i : b)  System.out.printf("%d ", i);  }      public static void main(String[] args) {  print("A", 11, 2, 31, 44);  print("B", 51, 35);  print("C", 64);  print("D");  int[] a = new int[] { 11, 2, 31, 44 };  print("E", a);  }  } |

실행 결과 출력

|  |
| --- |
| a=A, b.length=4 : 11 2 31 44  a=B, b.length=2 : 51 35  a=C, b.length=1 : 64  a=D, b.length=0 :  a=E, b.length=4 : 11 2 31 44 |

### VarArgs3.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.lecture04;  public class VarArgs3 {  static void print(int a, String... b) {  System.out.printf("\na=%d, b.length=%d : ", a, b.length);  for (String s : b)  System.out.printf("%s ", s);  }  public static void main(String[] args) {  print(1, "a", "bb", "c", "ddd", "ee");  print(2, "bb", "c");  print(3, "ddd");  print(4);  String[] a = new String[] { "a", "bb", "c", "ddd", "ee" };  print(5, a);  }  } |

실행 결과 출력

|  |
| --- |
| a=1, b.length=5 : a bb c ddd ee  a=2, b.length=2 : bb c  a=3, b.length=1 : ddd  a=4, b.length=0 :  a=5, b.length=5 : a bb c ddd ee |

# 유틸러티 클래스, 헬퍼 클래스

프로젝트를 구현할 때,

필수 입력 항목이 입력되었는지 검사할 일이 종종 있다.

예:

|  |
| --- |
| String s = textField.getText();  if (s == null || s.trim().length() == 0)  errorMessage = "내용을 입력하세요"; |

위와 같이 검사하기 위한 메소드들이 String 클래스에 있다면 편리할 것이다.

|  |
| --- |
| public static boolean isNullOrEmpty(String s); // s가 null 이거나, 빈 문자열이면 true를 리턴  public static boolean isNullOrBlank(Stirng s); // s가 null 이거나, 빈 문자열이거나,  // 공백이면 true를 리턴  public boolean isEmpty(Stirng s); // this 문자열이 빈 문자열이면 true를 리턴  public boolean isBlank(String s); // this 문자열이 빈 문자열이거나, 공백이면 true를 리턴 |

그런데 아쉽게도 위 메소들이 String 클래스에 없다.

String 클래스를 수정할 수도 없다.

## 자식 클래스

어떤 클래스에 메소드를 추가로 구현하려고 할 때,

만약 그 클래스를 수정할 수 없다면, 자식 클래스를 만들어서 메소드를 추가로 구현하면 된다.

|  |
| --- |
| class MyString extends String {  public static boolean isNullOrEmpty(String s) {  return s == null || s.length() == 0;  }  public static boolean isNullOrBlank(String s) {  return s == null || s.trim().length() == 0;  }  public boolean isEmpty() {  return this.length() == 0;  }  public boolean isBlank() {  return this.trim().length() == 0;  }  } |

그런데, String 클래스 대신 MyString 클래스를 사용하는 것은 불편하다.

|  |
| --- |
| MyString s = new MyString("hello"); |

언제나 위와 같이 MyString 객체를 생성해야 한다.

|  |
| --- |
| MyString s = "hello"; |

위와 같이 구현할 수 없다. "hello" 문자열은 String 클래스 객체이기 때문이다.

## 유틸러티 클래스

자식 클래스를 만들어서 메소드를 추가로 구현하는 것이, 어떤 이유 때문에 적절하지 않다면,

유틸러티 클래스를 만들어서 메소드를 추가로 구현한다.

|  |
| --- |
| class StringUtils {  public static boolean isNullOrEmpty(String s) {  return s == null || s.length() == 0;  }  public static boolean isNullOrBlank(String s) {  return s == null || s.trim().length() == 0;  }  public static boolean isEmpty(String s) {  return s.length() == 0;  }  public static boolean isBlank(String s) {  retusn s.trim().length() == 0;  }  } |

StringUtils 클래스의 모든 메소드는 static 메소드임에 주목하자.

StringUtils 클래스의 메소드는, StringUtils 클래스의 멤버 변수를 사용할 일이 없기 때문이다.

다음과 같이 사용한다.

|  |
| --- |
| String s = "hello";  if (StringUtils.isNullOrBlank(s) == false)  errorMessage = "내용을 입력하세요"; |

StringUtils 클래스와 같은 것들을, 유틸러티 클래스 혹은 헬퍼 클래스라고 부른다.

유틸러티 클래스의 이름은,

Strings 이거나 StringUtils 형태인 것이 관례이다.

유틸러티 클래스의 메소드는 거의 전부 static 메소드이다.

# Arrays 클래스

## Arrays 클래스

Arrays 클래스는 유틸러티 클래스이다.

배열을 다룰 때 유용한 메소드들이 Arrays 클래스에 구현되어 있다.

Arrays 클래스의 메소드는 대부분 static 메소드이다.

Arrays 클래스의 메소드는 this 객체에 어떤 작업을 하는 메소드가 아니고,

첫째 파라미터로 받은 배열에 대해서 어떤 작업을 하는 메소드이다.

그래서 Arrays 클래스의 메소드는 this 객체를 사용하지 않기 때문에 static 메소드이다.

Java 언어의 배열은 어떤 클래스 객체이고,

그 클래스는 Object 클래스로부터 equals 메소드, hashCode 메소드, toString 메소드를 상속 받았기 때문에,

배열에 대해서 위 메소드들을 사용할 수 있다.

그렇지만 Object 클래스로부터 상속 받은 메소드들을 사용하는 것은 바람직하지 않다.

예를 들어 Object 클래스의 equals 메소드는 equality를 비교하지 않고 identity를 비교하기 때문이다.

예:

|  |
| --- |
| int[] a = new int[] { 3, 4, 5 };  int[] b = new int[] { 3, 4, 5 };  System.out.println( a.equals(b) ); |

위 코드는 false를 출력한다.

그래서 배열 객체에 사용할 equals, hashCode, toString 메소드를 재정의 해야하지만,

Java 언어의 배열이 어떤 클래스의 객체인지, 그 클래스 이름이 감춰져 있어서,

그 클래스의 자식 클래스를 구현할 수 없다.

그래서 할 수 없이, Arrays 유틸러티 클래스에 위 메소드들을 구현해야 한다.

## Arrays 클래스의 메소드

아래 내용에서 T는 generic parameter 이다. 즉 T 위치에 어떤 타입이든 사용 가능하다.

|  |
| --- |
| static List<T> asList(T... a)  파라미터 값들로 List<T> 타입의 객체를 생성하여 리턴한다.  asList 메소드의 파라미터는 가변 파라미터이다.  예) List<String> stringList = Arrays.asList("one", "two", "three");  List<Person> personList = Arrays.asList(person1, person2, person3); |
| static int binarySearch(T[] a, T key)  배열 a에서 key 값을 찾아서 index를 리턴한다.  배열 a에 key 값이 없는 경우에는 key 값을 끼워 넣을 위치를 (index \* -1) - 1 값으로 리턴한다. |
| static T[] copyOf(T[] a, int newLength)  newLength 길이의 배열을 생성하고, a 배열의 값들을 새 배열에 복사하고, 새 배열을 리턴한다. |
| static T[] copyOfRange(T[] a, int from, in to)  배열 a에서 from index 위치에서 to index 위치까지의 값들로 구성된 새 배열을 생성하여 리턴한다. |
| static boolean equals(T[] a1, T[] a2)  배열 a1의 원소 각각에 대해서 equals 메소드를 호출하여  배열 a2에서 같은 위치(index)의 원소와 equality를 비교한다. 예: a1[i].equals(a2[i])  모든 원소의 equality가 같을 때 true를 리턴한다. |
| static boolean deepEquals(T[] a1, T[] a2)  Arrays.equals(T[] a1, T[] a2) 메소드와 거의 같은 일을 한다.  1차원 배열인 경우에는 equals 메소드를 사용하고,  다차원 배열인 경우에는 deepEquals 메소드를 사용해야 한다. |
| static void fill(T[] a, T value)  배열의 모든 원소에 value 값을 대입한다. |
| static void fill(T[] a, int from, int to, T value)  배열의 from index 에서 to index 까지 원소에 value 값을 대입한다. |
| static int hashCode(T[] a)  배열에 들어있는 객체들의 hashCode 값을 전부 고려하여 전체 hashCode 값을 계산하여 리턴한다.  Object 클래스에 int hashCode(); 메소드가 있다. 이 메소드는 this 객체의 hashCode 값을  계산하여 리턴한다. hash table에 객체를 저장할 때 이 hashCode 값이 사용된다. |
| static int deepHashCode(T[] a)  Arrays.hashCode(T[] a) 메소드와 거의 같은 일을 한다.  1차원 배열인 경우에는 hashCode 메소드를 사용하고,  다차원 배열인 경우에는 deepHashCode 메소드를 사용해야 한다. |
| static void sort(T[] a)  배열의 원소를 정렬한다. 배열의 원소의 타입 T는 기본 자료형이거나,  Comparable interface를 implements한 클래스 타입이어야 한다.  예를 들어 Java의 String 클래스나 Date 클래스는 Comparable interface를 implements 하였다.  값의 크기를 비교할 수 있는, Java 표준 라이브러리의 클래스들은 대부분 Comparable interface를  implements 하였다. |
| static void sort(T[] a, Comparator<T> comparator)  배열의 원소를 정렬한다. 정렬하기 위해서 배열의 원소를 비교할 때,  comparator의 compare 메소드를 사용한다. |
| static String toString(T[] a)  배열의 내용을 문자열로 변환해서 리턴한다. |
| static String deepToString(Object[] a)  다차원 배열의 내용을 문자열로 변환해서 리턴한다. |

binary search (이진 탐색)은 정렬된 배열에 대해서 탐색해야 한다.

정렬되지 않는 배열에 대해서 binary search 메소드를 호출하면 이상한 값이 리턴될 것이다 (에러가 발생할까?)

int[] a = int[] {1, 3, 5, 7, 9};

int index = Arrays.binarySearch(a, 6); // index 값은 -4

a 배열에 6을 끼워 넣을 위치의 인덱스는 3이다. (3 \* -1) - 1 = -4

## 예제 #1

### Arrays1.java

|  |  |
| --- | --- |
| 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  26  27  28  29  30  31  32  33 | package net.skhu.lecture04;  import java.util.Arrays;  import java.util.Random;  public class Arrays1 {    static Random random = new Random();    static int[] makeIntArray(int count) {  int[] a = new int[count];  for (int i = 0; i < count; ++i)  a[i] = random.nextInt(count);  return a;  }    public static void main(String[] args) {  int[] a = makeIntArray(20);  System.out.println(Arrays.toString(a));  Arrays.sort(a);  System.out.println(Arrays.toString(a));    int index = Arrays.binarySearch(a, 10);  System.out.printf("index of 10: %d\n", index);    int[] b = Arrays.copyOf(a, a.length);  System.out.printf("Arrays.equals(a, b) : %b\n", Arrays.equals(a, b));    System.out.printf("a.equals(b) : %b\n", a.equals(b));  System.out.println(a.toString());  }  } |

랜덤 정수 생성 방법.

|  |
| --- |
| Random random = new Random();  int i = random.nextInt(10); |

Random 클래스 객체를 사용해서 랜덤 정수를 생성한다.

random.nextInt(10) 메소드를 호출할 때마다 0 ~ 9 정수 중의 하나를 랜덤하게 골라서 리턴한다.

즉 Random 클래스의 int nextInt(int n) 메소드는 0 ~ n-1 정수 중의 하나를 랜덤하게 골라서 리턴한다.

(줄10) static int[] makeIntArray(int count)

파라미터로 주어진 count 크기의 int 배열을 생성하여 리턴한다.

배열에는 랜덤한 정수가 count 개 채워진다.

(줄26)int[] b = Arrays.copyOf(a, a.length);

a.length 길이의 새 배열을 생성해서, a 배열의 모든 원소를 새 배열에 복사하고, 그 새 배열에 대한 참조를 지역 변수 b에 대입한다.

(줄27) Arrays.equals(a, b)

a 배열과 b 배열의 내용 전체를 비교한다. 즉 equality를 비교한다.

두 배열의 내용이 동일하기 때문에 true를 리턴할 것이다.

(줄29) a.equals(b)

Java의 배열은 어떤 클래스의 객체이다.

즉 Java의 배열은 Object 클래스의 자식 클래스의 객체이다.

그래서 Object 클래스에서 상속된 equals 메소드를 사용할 수 있다.

Object 클래스의 equals 메소드는 identity를 비교하기 때문에 바람직하지 않다.

a 배열과 b 배열은, 내용은 동일하지만 서로 다른 두 개의 객체이기 때문에,

identity 비교 결과는 false 이다.

(줄30) Object 클래스에서 상속 받은 toString() 메소드를 호출한다.

## 예제 #2

### Arrays2.java

|  |  |
| --- | --- |
| 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  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package net.skhu.lecture04;  import java.util.Arrays;  import java.util.Random;  public class Arrays2 {    static Random random = new Random();    static String makeRandomString() {  StringBuilder builder = new StringBuilder();  for (int i=0; i < 4; ++i) {  int n = random.nextInt(26) + (int)'a';  builder.append((char)n);  }  return builder.toString();  }  static String[] makeStringArray(int count) {  String[] a = new String[count];  for (int i = 0; i < count; ++i)  a[i] = makeRandomString();  return a;  }  public static void main(String[] args) {  String[] a = makeStringArray(20);  System.out.println(Arrays.toString(a));  Arrays.sort(a);  System.out.println(Arrays.toString(a));    int index = Arrays.binarySearch(a, "nnn");  System.out.printf("index of \"nnn\": %d\n", index);    String[] b = Arrays.copyOf(a, a.length);  System.out.printf("Arrays.equals(a, b) : %b\n", Arrays.equals(a, b));    System.out.printf("a.equals(b) : %b\n", a.equals(b));  System.out.println(a.toString());  }  } |

(줄10) static String makeRandomString()

길이 3의 문자열(String) 객체를 생성하여 리턴한다.

3개의 문자는 랜덤하게 생성된다.

(줄13) (int)'a'

위 식은, 'a' 문자의 아스키 코드(ascii code) 값을 구한다.

'a' 문자의 아스키 코드 값은 97 이다.

따라서 위 식의 값은 97 이다.

문자가 컴퓨터에 저장될 때는 이진수 숫자로 변환되어 저장된다.

이 숫자 값이 아스키 코드 값이다.

(줄14) (char)n

위 식은, 정수 변수 n의 값과 아스크 코드가 일치하는 문자를 구한다.

예를 들어 변수 n의 값이 97 이라면 위 식의 값은 'a' 이고,

변수 n의 값이 98 이라면 위 식의 값은 'b' 이다.

(줄19) static String[] makeStringArray(int count)

파라미터로 주어진 count 크기의 String 배열을 생성하여 리턴한다.

배열에는 랜덤한 문자열 객체가 count 개 채워진다.

## 예제 #3

### Person.java

|  |  |
| --- | --- |
| 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  26  27  28  29  30  31 | package net.skhu.lecture04;  public class Person {  String name;  int age;    public Person(String name, int age) {  this.name = name;  this.age = age;  }    public String getName() {  return name;  }    public int getAge() {  return age;  }    @Override  public boolean equals(Object obj) {  if ((obj instanceof Person) == false) return false;  Person p = (Person)obj;  return (this.name == null ? p.name == null : this.name.equals(p.name)) && this.age == p.age;  }    @Override  public String toString() {  return String.format("Person{name=\"%s\", age=%d}", name, age);  }  } |

앞에서 구현했던 Person 클래스이다.

### PersonComparator.java

|  |  |
| --- | --- |
| 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  26  27  28  29 | package net.skhu.lecture04;  import java.util.Comparator;  public class PersonComparator implements Comparator<Person> {    public enum Compare { BY\_NAME\_ASC, BY\_NAME\_DESC, BY\_AGE\_ASC, BY\_AGE\_DESC };  private Compare compare;    public PersonComparator(Compare compare) {  this.compare = compare;  }  @Override  public int compare(Person p1, Person p2) {  if (p1 == p2) return 0;  if (p1 == null) return -1;  if (p2 == null) return 1;    switch (compare) {  case BY\_NAME\_ASC: return p1.getName().compareTo(p2.getName());  case BY\_NAME\_DESC: return p2.getName().compareTo(p1.getName());  case BY\_AGE\_ASC: return p1.getAge() - p2.getAge();  case BY\_AGE\_DESC: return p2.getAge() - p1.getAge();  }  return 0;  }  } |

정렬하기 위해서 Person 클래스의 객체를 비교하는 방법은 여러 개이다.

\* 이름 오름차순으로 정렬하기 위한 비교

\* 이름 내림차순으로 정렬하기 위한 비교

\* 나이 오름차순으로 정렬하기 위한 비교

\* 나이 내림차순으로 정렬하기 위한 비교

PersonComparator 클래스는 Person 객체를 비교하는 방법들을 구현한 클래스이다.

이렇게 비교하는 방법이 여러 개라면, 비교 방법들을 Person 클래스에 구현하는 것 보다,

별도의 클래스로 따로 구현하는 것이 바람직한 경우가 많다.

어떤 클래스의 비교 방법을 별도의 클래스로 따로 구현할 때,

Comparator interface를 implements 하여 구현하는 것이 바람직하다.

Java 표준 라이브러리에 이미 구현된 정렬 메소드나 이진 탐색 메소드에서 객체를 비교할 때,

Comparater interface를 구현한 클래스 객체를 요구하기 때문이다.

### Arrays3.java

|  |  |
| --- | --- |
| 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  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 | package net.skhu.lecture04;  import java.util.Arrays;  import java.util.Comparator;  import java.util.Random;  import net.skhu.lecture04.PersonComparator.Compare;  public class Arrays3 {  static Random random = new Random();    static String makeRandomString() {  StringBuilder builder = new StringBuilder();  for (int i=0; i < 4; ++i) {  int n = random.nextInt(26) + (int)'a';  builder.append((char)n);  }  return builder.toString();  }    static Person[] makePersonArray(int count) {  Person[] a = new Person[count];  for (int i = 0; i < count; ++i) {  String name = makeRandomString();  int age = random.nextInt(10) + 10;  a[i] = new Person(name, age);  }  return a;  }  public static void main(String[] args) {  Comparator<Person> personComparator = new PersonComparator(Compare.BY\_AGE\_DESC);    Person[] a = makePersonArray(10);  System.out.println(Arrays.toString(a));  Arrays.sort(a, personComparator);  System.out.println(Arrays.toString(a));    int index = Arrays.binarySearch(a, new Person("nnn", 10), personComparator);  System.out.printf("index of \"nnn\": %d\n", index);    Person[] b = Arrays.copyOf(a, a.length);  System.out.printf("Arrays.equals(a, b) : %b\n", Arrays.equals(a, b));    System.out.printf("a.equals(b) : %b\n", a.equals(b));  System.out.println(a.toString());  }  } |

(줄22) static Person[] makePersonArray(int count)

파라미터로 주어진 count 크기의 Person 배열을 생성하여 리턴한다.

배열에는 Person 객체가 count 개 채워진다.

Person 객체의 이름(name)과 나이(age)는 랜덤하게 생성한다.

## 예제 #4

### Arrays4.java

|  |  |
| --- | --- |
| 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  26  27  28  29  30  31  32  33  34 | package net.skhu.lecture04;  import java.util.Arrays;  import java.util.List;  public class Arrays4 {    static void printList(List<String> list) {  for (String s : list)  System.out.printf("%s ", s);  System.out.println();  }    public static void main(String[] args) {  String[] a = { "one", "two", "three" };  List<String> list1 = Arrays.asList("one", "two", "three");  List<String> list2 = Arrays.asList(a);    printList(list1);  printList(list2);    System.out.println(list1.toString());  System.out.println(list2.toString());    Object[] b = list1.toArray();  String[] c = (String[])list1.toArray();  System.out.printf("Arrays.equals(a, b): %b\n", Arrays.equals(a, b));  System.out.printf("Arrays.equals(a, c): %b\n", Arrays.equals(a, c));  System.out.printf("a == b: %b\n", a == b);  System.out.printf("a == c: %b\n", a == c);  System.out.printf("b == c: %b\n", b == c);  }  } |

(줄16) Arrays.asList("one", "two", "three")

파라미터로 주어진 문자열(String) 객체들이 채워진 목록(List<String>) 객체를 생성하여 리턴한다.

리턴되는 List<String> 타입의 객체의 사용법은 ArrayList<String> 타입의 객체의 사용법과 유사하다.

차이점은 목록의 길이가 고정이라는 점이다.

목록의 길이를 변경하는 메소드(add, remove)를 호출하면 에러(exception)가 발생한다.

Arrays.asList 메소드의 리턴 타입은 List<T> 타입이다. 여기서 T 는 generic parameter 이다.

목록에 실제로 add 되는 객체의 타입이 T 에 해당된다.

예를 들어 줄16에서 asList 메소드의 리턴 타입은 List<String> 타입이다.

여기서 List는 Java 표준 라이브러리에 들어있는 interface 이다. (java.util.List)

Array.asList 메소드가 리턴하는 것은 java.util.List 인터페이스를 implements한 어떤 클래스의 객체이다.

그런데 우리는 그 클래스가 구체적으로 무엇인지 모르고,

단지 그 클래스가 java.util.List 인터페이스를 implements 했다는 것만 알 수 있다.

따라서 그 클래스는 java.util.List 인터페이스의 모든 메소드를 재정의(override) 했을 것이 분명하다.

java.util.List 인터페이스를 implements한 대표적인 클래스는 java.util.ArrayList 클래스이다.

당연히 ArrayList 클래스도 java.util.List 인터페이스의 모든 메소드를 재정의 했다.

(줄19~20) 배열과 마찬가지로 List 타입의 객체도 for 루프로 탐색할 수 있다.

(줄22~23) Arrays.asList 메소드가 리턴하는 객체의 클래스 이름은 모르지만,

이 클래스에 toString() 메소드가 재정의 되어있다.

(줄25) List 인터페이스에 toArray() 메소드가 선언되어 있다.

이 메소드는 List 객체 내부에 들어있는 데이터 목록을 배열에 채워서 리턴한다.

새 배열 객체가 생성되어 리턴될 뿐, List 객체 내부에는 변화가 없다.

리턴되는 배열의 길이는 List 객체 내부에 들어있는 데이터 목록의 길이와 같다.

(줄25~26)

List 인터페이스의 toArray() 메소드는 호출될 때 마다 배열을 새로 생성해서 리턴한다.

배열에 채워진 값은 동일하므로 배열 b와 배열 c의 equality 비교는 true 이지만

배열은 매번 새로 생성되었기 때문에 identity 비교는 false 이다.

이 설명은 이해하지 못해도 괜찮다.

# 과제

아래 내용을 구현하라.

- Integer 객체가 여러 개 들어있는 배열을 생성하라.

예: Integer[] a = new Integer[] { 10, 3, 5, 2, 8 };

- 그 배열을 정렬하라.

- 그 배열에 어떤 값이 들어있는지 이진 탐색으로 확인하라.