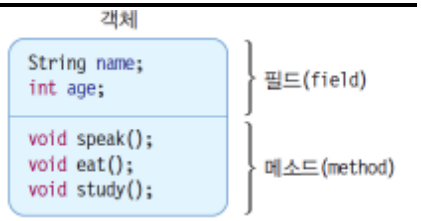
Class, Collections

514770-1
2017년 봄학기
3/22/2017
박경신

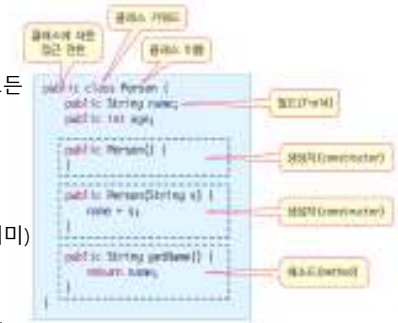
Class

- 클래스(Class)
  - 객체의 속성과 행위 선언
  - 객체의 설계도 혹은 틀
- 객체(Object)
  - 클래스의 틀로 찍어낸 실체
    - 메모리 공간을 갖는 구체적인 실체
    - 클래스를 구체화한 객체를 **인스턴스(instance)**라고 부름
    - 객체와 인스턴스는 같은 뜻으로 사용



클래스 구조

- 클래스 접근 권한, public
  - 다른 클래스들에서 이 클래스를 사용하거나 접근할 수 있음을 선언
- 클래스(class)
  - Person이라는 이름의 클래스 선언
  - 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 필드와 메소드 구현
- 필드(field)
  - 값을 저장할 멤버 변수 (혹은 필드)
  - 필드의 접근 지정자 public (다른 클래스의 메소드에서 호출할 수 있도록 공개한다는 의미)
- 메소드(method)
  - 메소드는 함수이며 객체의 행위를 구현
  - 메소드의 접근 지정자 public (다른 클래스의 메소드에서 호출할 수 있도록 공개한다는 의미)
- 생성자(constructor)
  - 클래스의 이름과 동일한 메소드
  - 클래스의 객체가 생성될 때만 호출되는 메소드



객체 생성 및 사용 예

- 객체 생성
  - new 키워드를 이용하여 생성
    - new는 객체의 생성자 호출
- 객체 생성 과정
  1. 객체에 대한 레퍼런스 변수 선언
  2. 객체 생성

접근 지정자

- 디폴트(default) 멤버
  - 같은 패키지 내의 다른 클래스만 접근 가능
- public 멤버
  - 패키지에 관계 없이 모든 클래스에서 접근 가능
- private 멤버
  - 클래스 내에서만 접근 가능
  - 상속 받은 하위 클래스에서도 접근 불가
- protected 멤버
  - 같은 패키지 내의 다른 모든 클래스에서 접근 가능
  - 상속 받은 하위 클래스는 다른 패키지에 있어도 접근 가능

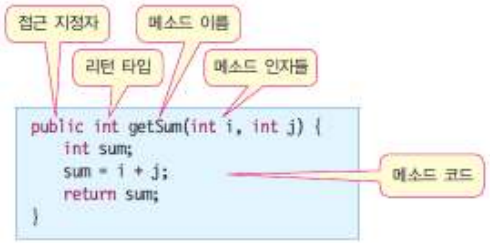
멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	○	×	○	○
다른 패키지의 클래스	×	×	×	○

Constructor

- 생성자(Constructor)
  - 객체가 생성될 때 초기화를 위해 실행되는 메소드
- 생성자의 특징
  - 생성자는 메소드
  - 생성자 이름은 클래스 이름과 동일
  - 생성자는 new를 통해 객체를 생성할 때만 호출됨
  - 생성자도 오버로딩하여 여러개 작성 가능
  - 생성자는 리턴 타입을 지정할 수 없음
  - 생성자는 하나 이상 선언되어야 함
    - 개발자가 생성자를 작성하지 않았으면 컴파일러에 의해 자동으로 기본 생성자가 선언됨
    - 기본 생성자를 디폴트 생성자(default constructor)라고도 함

Method

- 메소드
  - 메소드는 C/C++의 함수와 동일
  - 자바의 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
  - 접근 지정자
    - public. private, protected, 디폴트(접근 지정자 생략된 경우)
  - 리턴 타입
    - 메소드가 반환하는 값의 데이터 타입



get & set

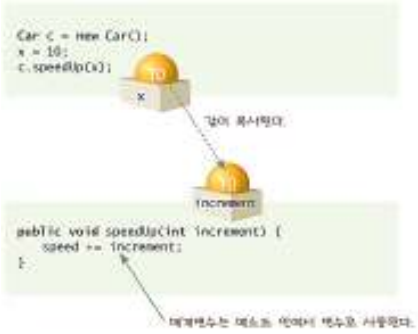
- 설정자(set)에서 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있음.
- 필요할 때마다 필드값을 계산하여 반환할 수 있음.
- 접근자(get)만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있음.

```
public void setSpeed(int s)
{
   if( s < 0 )
      speed = 0;
   else
      speed = s;
}
```

속도가 음수이면 0으로 만든다.

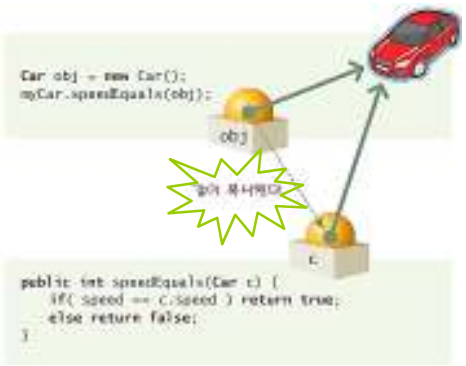
Parameter Passing – Primitive Type

- 인자 전달 (Parameter Passing)
  - **값에 의한 호출(call by value)**
  - 기본 타입의 값을 전달하는 경우
    - 값이 복사되어 전달
    - 메소드의 매개 변수가 변경되어도 호출한 실인자 값은 변경되지 않음

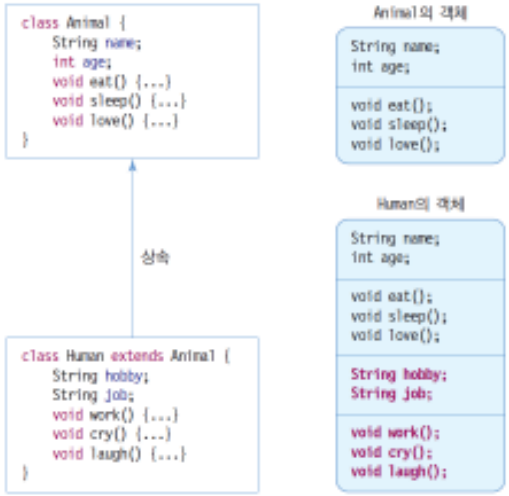


Parameter Passing – Reference Type

- 인자 전달 (Parameter Passing)
  - 객체(object) 혹은 배열(array)을 전달하는 경우
    - 객체나 배열의 레퍼런스(reference)만 전달
      - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
    - 메소드의 매개 변수와 호출한 실인자가 객체나 배열을 공유하게 됨



Inheritance



- 상속(Inheritance)
  - 상위 클래스의 특성을 하위 클래스가 물려받음
    - 상위 클래스 : 수퍼 클래스, 하위 클래스 : 서브 클래스
  - 서브 클래스
    - 수퍼 클래스 코드의 재사용
    - 새로운 특성 추가 가능
  - 자바는 클래스 다중 상속 없음
    - 인터페이스를 통해 다중 상속과 같은 효과 얻음

상속과 접근 지정자

- 자바의 접근 지정자 (public, protected, default, private)
  - 상속 관계에서 주의할 접근 지정자는 private와 protected
- 슈퍼 클래스의 private 멤버
  - 슈퍼 클래스의 private 멤버는 다른 모든 클래스에 접근 불허
- 슈퍼 클래스의 protected 멤버
  - 같은 패키지 내의 모든 클래스 접근 허용
  - 동일 패키지 여부와 상관없이 서브 클래스에서 슈퍼 클래스의 protected 멤버 접근 가능

슈퍼 클래스 멤버에 접근하는 클래스 종류	슈퍼 클래스 멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	○	×	○	○
다른 패키지의 클래스	×	×	×	○
같은 패키지의 서브 클래스	○	×	○	○
다른 패키지의 서브 클래스	×	×	○	○

(○는 접근 가능함을, ×는 접근이 불가함을 뜻함)

this

- this 란?
  - 현재 실행되는 메소드가 속한 객체에 대한 레퍼런스
    - 컴파일러에 의해 자동 선언 : 별도로 선언할 필요 없음

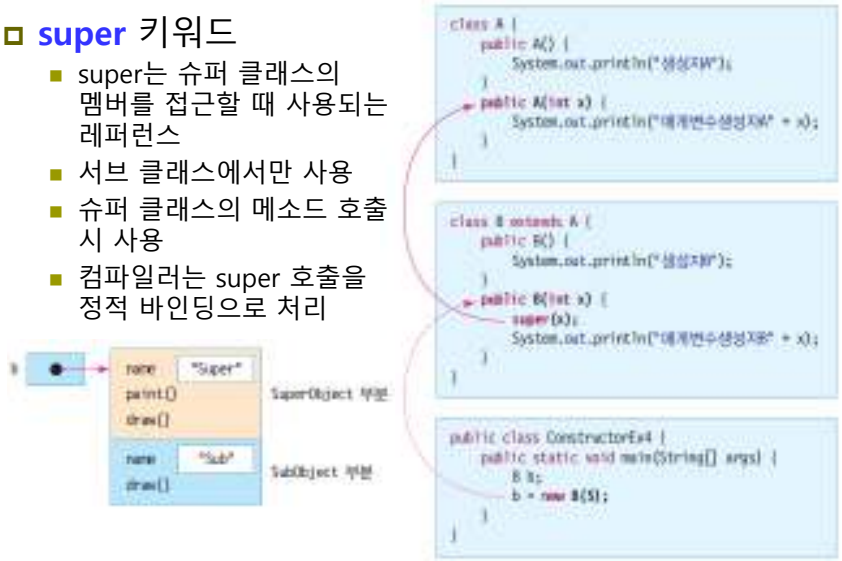
```
class Samp {
  int id;
  public Samp(int x) { id = x; }
  public void set(int x) { id = x; }
  public int get() {return id; }
}
```

=

```
class Samp {
  int id;
  public Samp(int x) { this.id = x; }
  public void set(int x) { this.id = x; }
  public int get() {return id; }
}
```

super

- super 키워드
  - super는 슈퍼 클래스의 멤버를 접근할 때 사용되는 레퍼런스
  - 서브 클래스에서만 사용
  - 슈퍼 클래스의 메소드 호출 시 사용
  - 컴파일러는 super 호출을 정적 바인딩으로 처리



Method Overloading

- 메소드 오버로딩(Overloading)
  - 한 클래스 내에서 두 개 이상의 이름이 같은 메소드 작성
    - 메소드 이름이 동일하여야 함
    - 매개 변수의 개수가 서로 다르거나, 타입이 서로 달라야 함
    - 리턴 타입은 오버로딩과 관련 없음

```
// 메소드 오버로딩이 성공한 사례
class MethodOverloading {
  public int getSum(int i, int j) {
    return i + j;
  }
  public int getSum(int i, int j, int k) {
    return i + j + k;
  }
  public double getSum(double i, double j) {
    return i + j;
  }
}
```

```
// 메소드 오버로딩이 실패한 사례
class MethodOverloadingFail {
  public int getSum(int i, int j) {
    return i + j;
  }
  public double getSum(int i, int j) {
    return (double)(i + j);
  }
}
```

Method Overriding

- 메소드 오버라이딩(Method Overriding)
  - 슈퍼 클래스의 메소드를 서브 클래스에서 재정의
    - 슈퍼 클래스의 메소드 이름, 메소드 인자 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
      - 이 중 하나라도 다르면 메소드 오버라이딩 실패
  - 동적 바인딩 발생
    - 서브 클래스에 오버라이딩된 메소드가 무조건 실행되도록 동적 바인딩 됨

static vs. non-static

- non-static 멤버의 특성
  - 공간적 - 멤버들은 객체마다 독립적으로 별도 존재
    - 인스턴스 멤버라고도 부름
  - 시간적 - 필드와 메소드는 객체 생성 후 비로소 사용 가능
  - 비공유의 특성 - 멤버들은 여러 객체에 의해 공유되지 않고 배타적
- static 멤버란?
  - 객체를 생성하지 않고 사용가능
  - 클래스당 하나만 생성됨
    - 클래스 멤버라고도 부름
    - 객체마다 생기는 것이 아님
  - 특성
    - 공간적 특성 - static 멤버들은 클래스 당 하나만 생성.
    - 시간적 특성 - static 멤버들은 클래스가 로딩될 때 공간 할당.
    - 공유의 특성 - static 멤버들은 동일한 클래스의 모든 객체에 의해 공유

```
class StaticSample {
  int n; // non-static 필드
  void g() {...} // non-static 메소드

  static int m; // static 필드
  static void f() {...} // static 메소드
}
```

final

- final 클래스 - 더 이상 클래스 상속 불가능

```
final class FinalClass { .....
}
class DerivedClass extends FinalClass { // 컴파일 오류
      .....
}
```

- final 메소드 - 더 이상 오버라이딩 불가능

```
public class SuperClass {
      protected final int finalMethod() { ... }
}
class DerivedClass extends SuperClass {
      protected int finalMethod() { ... } // 컴파일 오류, 오버라이딩 할 수 없음
}
```

- final 필드 – 상수 정의

```
public class FinalFieldClass {
      static final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정
}
```

객체의 타입 변환

- 업캐스팅(upcasting)
  - 프로그램에서 이루어지는 자동 타입 변환
  - 서브 클래스의 레퍼런스 값을 슈퍼 클래스 레퍼런스에 대입
    - 슈퍼 클래스 레퍼런스가 서브 클래스 객체를 가리키게 되는 현상
    - 객체 내에 있는 모든 멤버를 접근할 수 없고 슈퍼 클래스의 멤버만 접근 가능

```
class Person {
}

class Student extends Person {
}
...

Student s = new Student();
Person p = s; // 업캐스팅, 자동타입변환
```

객체의 타입 변환

- 다운캐스팅(downcasting)
  - 슈퍼 클래스 레퍼런스를 서브 클래스 레퍼런스에 대입
  - 업캐스팅된 것을 다시 원래대로 되돌리는 것
  - 명시적으로 타입 지정

```
class Person {
}
class Student extends Person {
}
...

Student s = (Student)p; // 다운캐스팅, 강제타입변환
```

instanceof

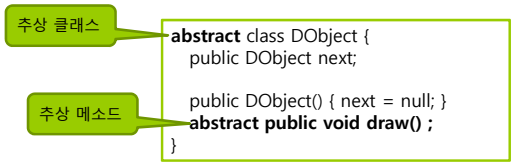
- 업캐스팅된 레퍼런스로는 객체의 진짜 타입을 구분하기 어려움
  - 슈퍼 클래스는 여러 서브 클래스에 상속되기 때문
    - 슈퍼 클래스 레퍼런스로 서브 클래스 객체를 가리킬 수 있음
- **instanceof** 연산자
  - instanceof 연산자
    - 레퍼런스가 가리키는 객체의 진짜 타입 식별
  - 사용법

객체레퍼런스 **instanceof** 클래스타입

연산의 결과 : true/false의 불린 값

Abstract Class & Method

- 추상 메소드(abstract method)
  - 선언되어 있으나 구현되어 있지 않은 메소드
    - **abstract** 키워드로 선언
    - ex) public abstract int getValue();
  - 추상 메소드는 서브 클래스에서 오버라이딩하여 구현
- 추상 클래스(abstract class)
  1. 추상 메소드를 하나라도 가진 클래스
     - 클래스 앞에 반드시 abstract라고 선언해야 함
  2. 추상 메소드가 하나도 없지만 클래스 앞에 abstract로 선언한 경우



추상 클래스의 상속

- 추상 클래스의 상속 2 가지 경우
  - 추상 클래스의 단순 상속
    - 추상 클래스를 상속받아, 추상 메소드를 구현하지 않으면 서브 클래스도 추상 클래스 됨
    - 서브 클래스도 abstract로 선언해야 함

```
abstract class DObject { // 추상 클래스
  public DObject next;
  public DObject() { next = null; }
  abstract public void draw(); // 추상 메소드
}
abstract class Line extends DObject { // draw()를 구현하지 않았기 때문에 추상 클래스
  public String toString() { return "Line";}
}
```

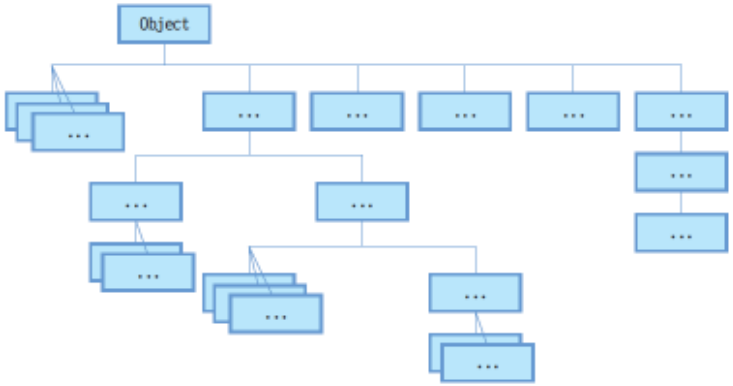
  - 추상 클래스 구현 상속
    - 서브 클래스에서 슈퍼 클래스의 추상 메소드 구현(오버라이딩)
    - 서브 클래스는 추상 클래스 아님

Polymorphism

- 다형성(Polymorphism)
  - 다형성(polymorphism)이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 각 객체의 타입에 따라서 서로 다른 동작을 하는 것(dynamic binding)
  - 자바의 다형성 사례
    - 슈퍼 클래스의 메소드를 서브 클래스마다 다르게 구현하는 메소드 오버라이딩(overriding)

자바의 클래스 계층 구조

자바에서는 모든 클래스는 반드시 java.lang.Object 클래스를 자동으로 상속받는다.



Object의 메소드

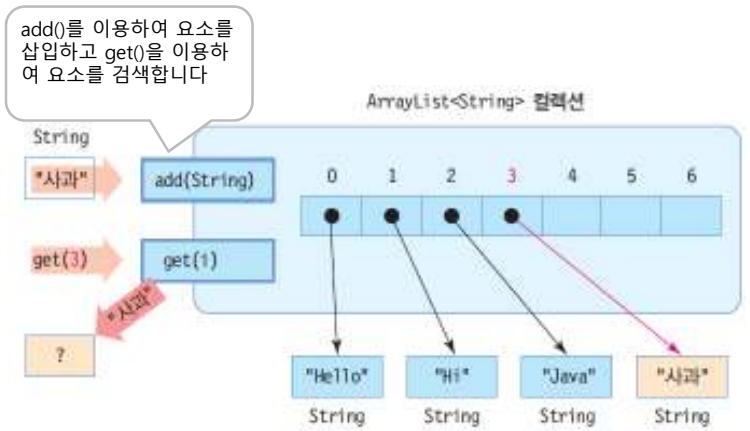
메소드	설명
protected Object clone()	현 객체와 똑같은 객체를 만들어 리턴
boolean equals(Object obj)	obj가 가리키는 객체와 현재 객체가 비교하여 같으면 true 리턴
Class getClass()	현 객체의 런타임 클래스를 리턴
int hashCode()	현 객체에 대한 해시 코드 값 리턴
String toString()	현 객체에 대한 스트링 표현을 리턴
void notify()	현 객체에 대해 대기하고 있는 하나의 스레드를 깨운다.
void notifyAll()	현 객체에 대해 대기하고 있는 모든 스레드를 깨운다.
void wait()	다른 스레드가 해올 때까지 현재 스레드를 대기하게 한다.

ArrayList<E>

- ArrayList<E>의 특성
  - java.util.ArrayList, 가변 크기 배열을 구현한 클래스
    - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
  - ArrayList에 삽입 가능한 것
    - 객체, null
    - 기본 타입은 박싱/언박싱으로 Wrapper 객체로 만들어 저장
  - ArrayList에 객체 삽입/삭제
    - 리스트의 맨 뒤에 객체 추가
    - 리스트의 중간에 객체 삽입
    - 임의의 위치에 있는 객체 삭제 가능
  - 벡터와 달리 스레드 동기화 기능 없음
    - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화되지 않음
    - 개발자가 스레드 동기화 코드 작성

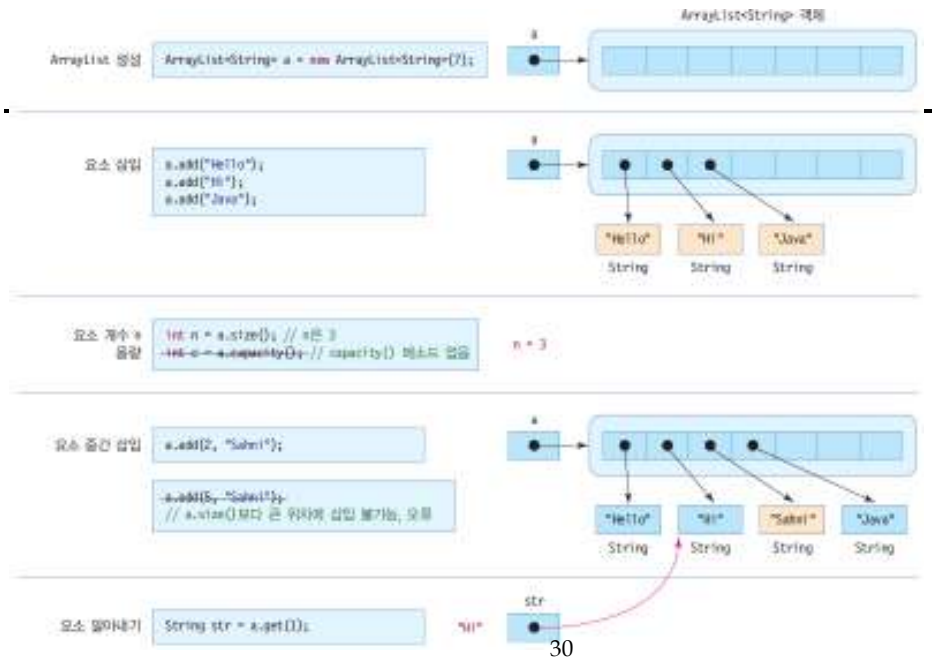
ArrayList<String> 컬렉션의 내부 구성

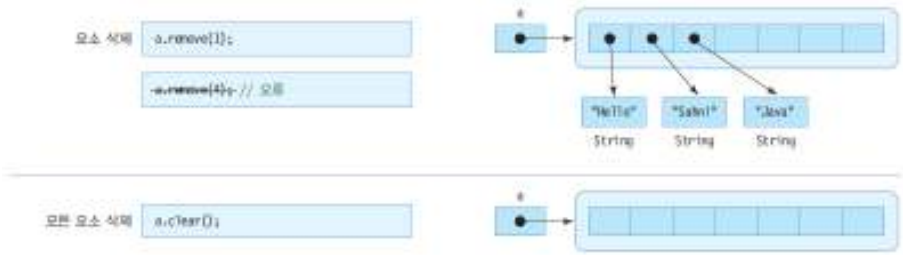
```
ArrayList<String> = new ArrayList<String>();
```



ArrayList<E> 클래스의 주요 메소드

메소드	설명
boolean add(E element)	ArrayList의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index에 지정된 element 삽입
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 ArrayList의 맨 뒤에 추가
void clear()	ArrayList의 모든 요소 삭제
boolean contains(Object o)	ArrayList가 지정된 객체를 포함하고 있으면 true 리턴
E elementAt(int index)	index 인덱스의 요소 리턴
E get(int index)	index 인덱스의 요소 리턴
int indexOf(Object o)	o와 같은 첫 번째 요소의 인덱스 리턴. 없으면 -1 리턴
boolean isEmpty()	ArrayList가 비어 있으면 true 리턴
E remove(int index)	index 인덱스의 요소 삭제
boolean remove(Object o)	o와 같은 첫 번째 요소를 ArrayList에서 삭제
int size()	ArrayList가 포함하는 요소의 개수 리턴
Object[] toArray()	ArrayList의 모든 요소를 포함하는 배열 리턴





ArrayList에 문자열을 달기

키보드로 문자열을 입력 받아 ArrayList에 삽입하고 가장 긴 이름을 출력하라.

```
import java.util.*;

public class ArrayListEx {
  public static void main(String[] args) {
    // 문자열만 삽입가능한 ArrayList 컬렉션 생성
    ArrayList<String> a = new ArrayList<String>();

    // 키보드로부터 4개의 이름 입력받아 ArrayList에 삽입
    Scanner scanner = new Scanner(System.in);
    for(int i=0; i<4; i++) {
      System.out.print("이름을 입력하세요>>");
      String s = scanner.next(); // 키보드로부터 이름 입력
      a.add(s); // ArrayList 컬렉션에 삽입
    }
```