SECTION 1. 상속

1. 자바 클래스의 상속은 단일 상속이다. > 다중 상속은 객체의 명확성이 떨어진다!
2. 특정 클래스가 가지는 일부 속성과 기능을 다른 새로운 클래스에게 제공하기 위해 맺는 클래스간의 관계를 말한다.

Base class = super class = parent class

* 상속을 주기 위해 준비된 특정 클래스

Derivation class = sub class = child class

* 특정 클래스로부터 상속을 받아 새롭게 정의되는 클래스

1. 클래스 상속은 객체의 재사용과 코드의 간결성을 제공해준다.
2. Overriding: 메서드 재정의라고 불리며 메서드 명이 같아야 하고 인자가 같아야 한다.

리턴형도 같아야 한다. super 클래스의 메서드보다 sub 클래스의 메서드 접근제한이 동일하거나 더 넓어야한다.

: 부모 클래스가 가지는 메서드를 자식 클래스에서 똑같이 재정의하는 것.

: 부모 클래스의 기능을 변경하거나 필요에 따라 확장을 필요할 때 사용한다.

super와 super()

1. 개념

: 특정 객체 내에서 자기 자신의 객체를 참조할 수 있는 유일한 참조변수인 this와 함께 객체를 참조할 수 있는 참조변수이다.

: 현재 객체의 바로 위의 super 클래스(부모 클래스)를 참조한다.

: 오버라이딩으로 멤버 은폐가 되어 super 클래스의 일부를 사용할 수 없느 ㄴ상태가 되었을 때 은폐된 것을 sub 클래스에서 호출하고자 할때 또는 super 클래스 자체를 참조하고자 할 때 사용하는 것이 super라는 예약어이다.

SECTION 2. final 예약어

1. 개념

: 프로그램이 끝날 때까지 값을 변경하지 못하는 Constant화가 된다.

: 상수는 static 예약어와 함께 정적화하여 특정 클래스의 객체가 여러 개 생성되더라도 오로지 1개만 만들어지도록 하는 것이 보통이다.

: 메서드에 final을적용하면 상속관계에 따른 오버라이딩으로의 확장을 할 수 없게 된다.

: 클래스에 final을 적용하면 더 이상 상속을 허용하지 않은 종단 클래스가 된다.

SECTION 3. 추상화

1. 개념

: 변수와 메소드를 encapsulation하고 information hidihng 까지 한 것

: 구체적인 개념으로부터 공통된 부분들만 추려내어 일반화 할 수 있도록 하는 것

|  |  |
| --- | --- |
| 구성 | 사용 예 |
| [접근제한] abstract void [메서드 명](); | public abstract void absTest(); |
| [접근제한] abstract class [클래스 명](); | public abstract class AbsEx(); |

2. 추상 클래스의 상속 관계

: 일반 클래스간의 상속과 유사하지만 추상 클래스간의 상속에서는 상속받은 추상 메서드들을 꼭 재정의 할 필요는 없다.

: 특정 클래스에서 일부 기능은 공통된 기능이지만일부 기능이 상속받는 각 클래스들마다 특징적인 변화가 있을 때 상속에 의해 다른 부분만 오버라이딩하여 사용한다.

3. 정리

: 추상 크래스는 공통된 부분만 구현하고 공통되지 않은 부분들은 abstract라는 예약어를 사용한 미완성된 클래스이므로 다른 클래스로 상속되어 미완성된 부분들을 재정의하여 완성시킨 뒤 사용할 수 있다.

: 추상클래스의 필요성이라 함은 여러 개의 클래스들이 상당수의 공통된 점을 가지고 있고 소수의 다른 점들이 있을 때 효율적인 것이다.

SECTION 4. 인터페이스

1. 정의

: 서비스 요청에 따른 중계자 역할

: 명시한 인터페이스가 가지는 추상 메서드들은 구현받은 클래스에서 하나도 빠짐없이 오버라이딩 해야한다.

|  |  |
| --- | --- |
|  | 사용 예 |
|  | [접근제한] interface 인터페이스명 {  상수;  추상메서드;  } |

2. 인터페이스 간의 상속

|  |  |
| --- | --- |
|  | 사용 예 |
|  | [접근제한] interface 인터페이스명 extends 부모\_인터페이스명1, 부모\_인터페이스명2... {  상수;  추상메서드;  } |

: 상속을 받은 자식 인터페이스를 구현하는 일반 클래스에서 부모 인터페이스와 자식 인터페이스의 추상 메서드들을 모두 오버라이딩 해야한다.

3. 인터페이스의 데폴트 메서드가 중복되는 경우

: 구현 코드를 가지고 인스턴스 생성된 경우만 호출되는 디폴트 메서드의 경우 두 개의 인터페이스에서 중복되면 구현하는 클래스에서 반드시 재정의 해야 한다.

4. InstanceOf 예약어

: 예약어를 중심으로 해서 왼쪽의 객체가 오른쪽에 명시한 클래스로부터 생성이 되었는지에 대한 형식을 비교하는 미리 정의된 연산자다.

|  |  |
| --- | --- |
|  | 사용 예 |
|  | String res = null;  if(it1 instanceOf Inter2)  res = “yes”;  else  res = “no”; |

5. Object 클래스

: 클래스들의 계층구조에서 루트이다. 즉, 모든 클래스들의 super 클래스이다.

SECTION 5. 열거형

1. 개념

: 상수를 가지고 생성되는 객체들을 한 곳에 모아둔 하나의 묶음.

: 열거형은 하나의 클래스를 의미한다.

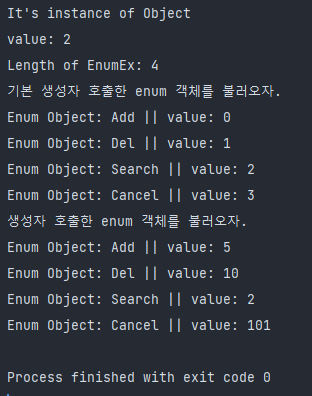
: 각 상수들은 하나의 객체로 인식되고 있어 0부터 1씩 증가하면서 각각의 열거형 객체를 생성하여 객체의 값으로 대입하게 된다.

|  |  |
| --- | --- |
|  | 사용 예 |
|  | [접근제한] enum 열거형\_이름{  상수1,  상수2,  ......  상수n  } |

: 열거형 내에 정의한 객체들이 열거형 자신의 형태로 객체가 생성되어 배열로 관리됨을 알 수 있다. 그리고 그 객체 하나 하나가 기억하고 있는 실제 정수값은 ordinal() 메서드를 통해 확인할 수 있다. 또한 아래 예시와 같이 프로그래머가 원하는 값으로 설정할 수도 있다.







SECTION 6. 내부클래스

1. 내부 클래스 정의

: 내부 클래스란 특정 클래스 안에 또 다른 클래스가 정의되는 것을 의미한다.

: 내부 클래스는 외부 클래스의 모든 멤버들을 마치 자신의 멤버처럼 사용할 수 있다.

: static 내부 클래스를 제외하고는 다른 내부 클래스는 항상 외부 클래스를 통해야 생성할 수 있다.

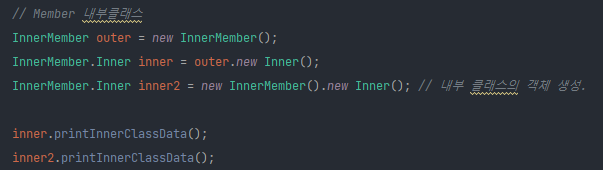
: 컴파일 후 확인해볼때 class 파일들 이름 중간에 ($)가 있는 것은 모두 내부 클래스의 바이트코드이다. Anonymous 내부 클래스는 클래스 이름이 없어 컴파일러가 임의로 숫자와 기호를 붙여 컴파일 후 바이트 코드 생성한다.

|  |  |
| --- | --- |
| 종류 | 설명 |
| Member | 멤버 변수나 멤버 메서들들과 같이 클래스가 정의된 경우에 사용한다. |
| Local | 특정한 메서드 내에 클래스가 정의된 경우에 사용한다. |
| Static | static 변수(클래스 변수)와 같이 클래스가 static으로 선언된 경우에 사용한다. |
| Anonymous | 참조할 수 있는 이름이 없는 경우에 사용한다. |

■ Member 내부 클래스

: 객체를 생성해야만 사용할 수 있는 멤버들과 같은 위치에 정의되는 클래스를 말한다.

: 내부 클래스를 생성하려면 외부 클래스의 객체를 생성한 후에 생성할 수 있다.



\* InnerMember 내부에 있는 Inner 객체형으로 변수 inner2를 선언하고 InnerMember를 먼저 생성한 후 생성된 객체를 이용해서 바로 내부에 있는 Inner 객체를 생성하여 생성된 참조변수를 inner2에게 전달한다.

■ Local 내부 클래스

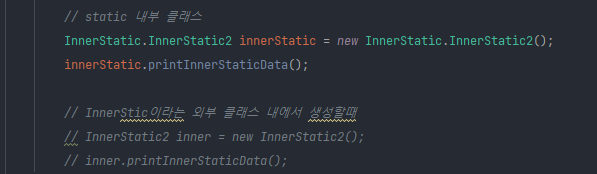
: 특정 메서드 안에서 정의되는 클래스를 말한다. 다시 말해, 특정 메서드 안에서 선언되는 지역 변수와 같은 것이다.

: 메서드가 호출될 때 생성할 수 있으며 메서드의 수행력이 끝나면 지역변수와 같이 자동 소멸된다.



■ static 내부 클래스

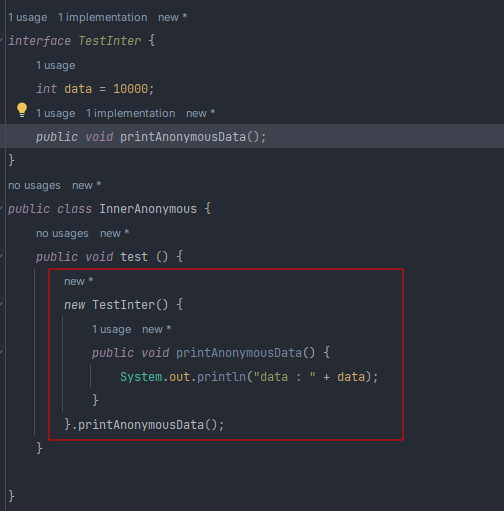
: 내부 클래스 안에 static 변수를 가지고 잇을 때 내부 클래스를 static으로 선언해야하는데 이 경우 사용한다.





■ Anonymous 내부 클래스

: 한 번만 사용하고 버려지는 객체를 사용할 때 유용한 내부 클래스다. 예를 들어, 프로그램을 종료할 때 꼭 수행해야 할객체가 있다면 한 번 수행 후 프로그램이 종료되므로 클래스가 더 이상 필요없게 된다. 이렇게 단 한번만 사용되는 객체들은 익명 내부 클래스를 사용하면 매우 유용하다.

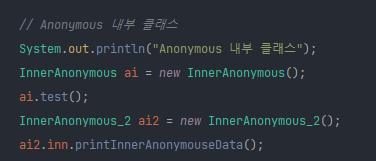


\* 메서드 수행. 인터페이스 TestInter를 생성한다. 하지만 인터페이스는 instance를 가지지 못한다. 이유는 인터페이스 안에는 완성되지 않은 추상 메서드들로 인해 자생력이 없다. 그런데 코드 내용을 살펴보면 마치 인터페이스를 명시적으로 생성을 하는 것 같은 생각이 든다. 하지만 사실 그런 것은 아니다. 이는 TestInter라는 인터페이스를 구현하는 내부 클래스가 자동으로 정의되는 것이다. 그리고 내부에서는 인터페이스에 정의된 추상 메서드들의 재정의를 통해서 객체를 완성할 수 있도록 해야한다. 이렇게 생성된 내부 클래스의 객체를 참조할 수 있는 참조변수가 없으므로 해서 익명 내부 클래스라고 하는 것이다.

\* 아래는 내부 클래스의 이름은 존재하지 않지만 참조할 수 있는 참조 변수의 이름이 잇는 경우이다. 참조변수가 잇으므로 정의된 영역 안에서는 얼마든지 참조변수를 통해 참조할 수 있다.



=========================== main ================================



**SECTION 7. 제네릭**

자바에서 제네릭이란 데이터의 타입을 일반화(Generalize)하는 것을 의미한다.

제네릭은 컬렉션(자료구조), 즉 객체들을 저장하는 구조적인 성격을 보강하기 위해 제공되는 것이다. 제네릭을 통해 개발자가 특정 컬렉션에 원하는 객체 자료형을 명시하여 실행하기 전에 컴파일 단계에서 특정 컬렉션에 대입되는 객체가 명시된 객체가 아니면 절대 저장이 불가능하게 할 수 있다.

제네릭은 클래스나 메소드에서 사용할 내부 데이터 타입을 컴파일 시에 미리 지정하는 방법이다.

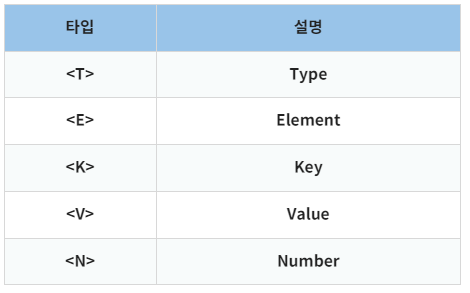
이렇게 컴파일 시에 미리 type check를 수행하면 아래와 같은 장점을 갖는다.

* 클래스나 메소드 내부에서 사용되는 객체의 타입 안정성을 높일 수 있다.
* 반환값에 대한 타입 변환 및 타입 검사에 들어가는 노력을 줄일 수 있다.

JDK 1.5 이전에서는 여러 타입을 사용하는 대부분의 클래스나 메소드에서 인수나 반환값으로 Object타입을 사용했다. 하지만 이 경우에는 반환된 Object 객체를 다시 원하는 타입으로 타입 변환해야 하며, 이때 오류가 발생할 가능성도 존재한다. 하지만 JDK 1.5 부터 도입된 제네릭을 사용하면 컴파일 시에 미리 타입이 정해지므로 타입 검사나 타입 변환과 같은 번거로운 작업을 생략할 수 있다.

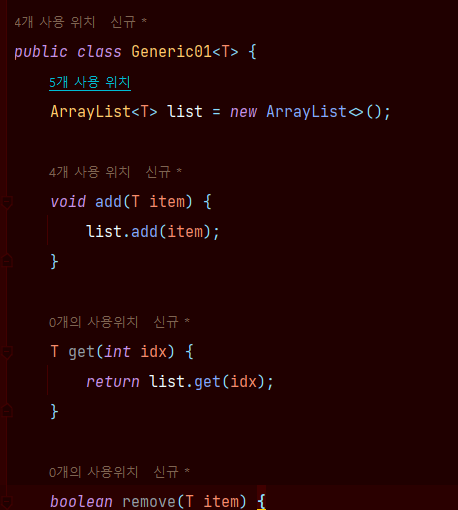
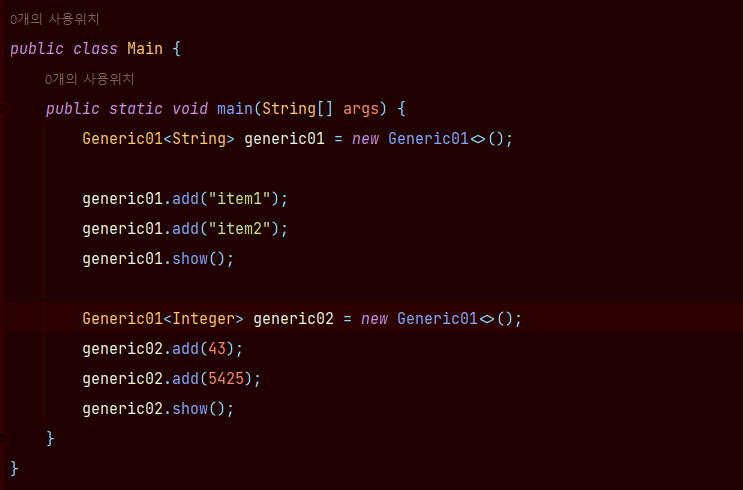
1. 제네릭 타입

■ 한 글자로 된 영문대문자

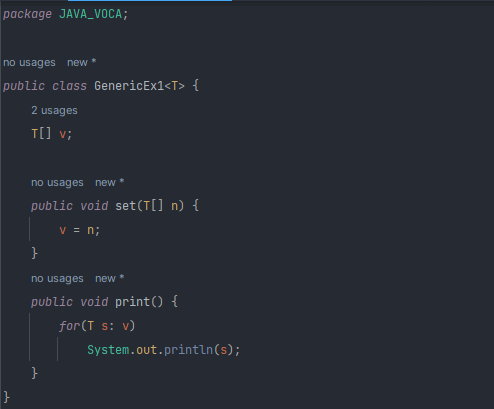


임의의 참조형 타입이라고 하고 위 코드에서 <T>와 같은 것을 말한다. 반드시 위 설명대로 쓰지 않아도 되고 여러 개의 타입 변수는 쉼표로 구분하여 명시할 수 있다. 타입 변수는 클래스에서 뿐만 아니라 메소드의 매개변수나 반환 값으로도 사용할 수 있다.

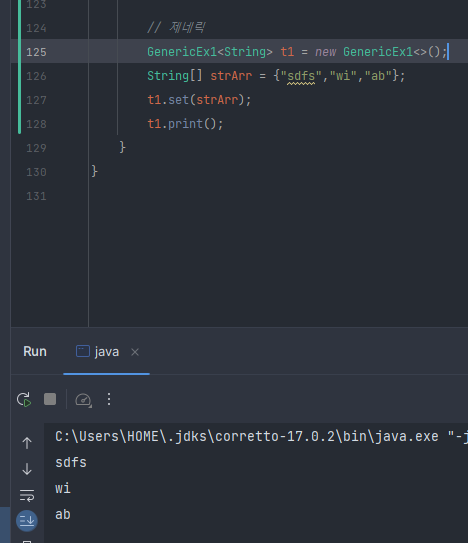
위와 같이 선언된 제네릭 클래스를 생성할 때에는 타입 변수 자리에 사용할 실제 타입을 명시해야 한다. JAVA에서 타입 변수 자리에 사용할 실제 타입을 명시할 때 기본 타입을 바로 사용할 수는 없다. 이때는 Wrapper class를 사용해야 한다.



2. 사용자 정의 제네릭 클래스



3. 제네릭 타입 사용하기



: 메서드를 호출할 때 사용하는 인자가 String형 배열이 아닌 다른 타입이라면 “~~ is not applicable for the arguments~~” 에러가 나온다. 이를 해결하기 위한 것이 와일드카드이다.

4. 제네릭의 제거 시기

자바 코드에서 선언되고 사용된 제네릭 타입은 컴파일 시 컴파일러에 의해 자동으로 검사되어 타입 변환된다. 그리고서 코드 내의 모든 제네릭 타입은 제거되어 컴파일된 class 파일에는 어떤 제네릭 타입도 포함되지 않게 된다.

이런 식으로 동작하는 이유는 제네릭을 사용하지 않는 코드와의 호환성을 유지하기 위해서이다.

5. 타입 변수의 제한

제네릭은 타입 변수를 사용하여 타입을 제한하는데 이때 extends 키워드를 사용하면 타입 변수에 특정 타입만 사용하도록 제한할 수 있다.

class list<T extends Aclas> { … }

위와 같이 클래스의 타입 변수에 제한을 걸어 놓으면 클래스 내부에서 사용된 모든 타입 변수에 제한이 걸린다. 이때에는 클래스가 아닌 인터페이스를 구현할 경우에도 implements가 아닌 extends 키워드를 사용해야 한다.

Class list<T extends Aclas & Ainter> { … }

6. 제네릭 메소드

제네릭 메소드란 메소드의 선언부에 타입 변수를 사용한 메소드를 의미한다.

Public static <T> void sort(…) {…}

이때 타입 변수의 선언은 메소드 선언부에서 반환 타입 바로 앞에 위치한다.

class list<T> {

…

public static <T> void sort(List<T> list, Comparator<? super T comp) {

… }

…

}

위 코드에서 제네릭 클래스에 정의된 타입변수 T와 제네릭 메소드에 정의된 타입변수 T는 별개의 것이다.

7. 와일드카드

이름에 제한을 두지 않음을 표현하는데 사용되는 기호이다.

자바의 제네릭에서는 물음표(?) 기호를 사용하여 이러한 와일드카드를 사용할 수 있다.

<?> // 타입 변수에 모든 타입을 사용할 수 있음.

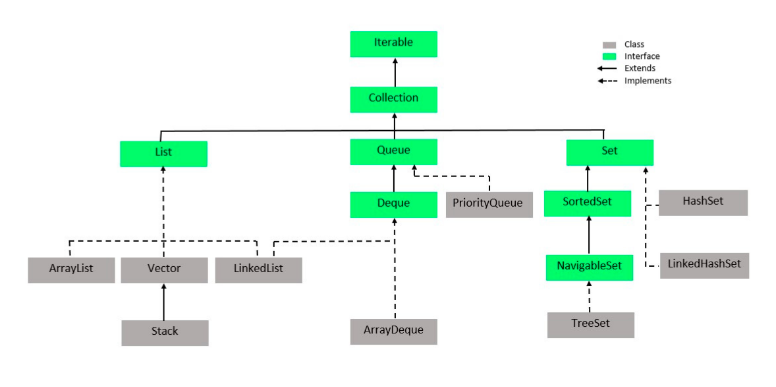
<? extends T> // T 타입과 T 타입을 상속받는 자손 클래스 타입만을 사용할 수 있음.

<? super T> // T 타입과 T 타입이 상속받은 조상 클래스 타입만을 사용할 수 있음

**SECTION 8. 자바 컬렉션 프레임워크**

컬렉션은 다수의 요소를 하나의 그룹으로 묶어 효율적으로 저장하고, 관리할 수 있는 기능을 제공하는 일종의 컨테이너이다. 다수의 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공하는 클래스의 집합을 의미한다.

배열은 크기가 고정되어 있는데에 반해, 컬렉션 프레임워크는 가변적인 크기를 갖는 등의 특징을 갖는다. 또한 데이터 삽입, 탐색, 정렬 등 편리한 API를 다수 제공한다. 이런 이점으로 개발자는 배열보다는 적절한 컬렉션 클래스를 선택해 사용하는 것이 권장된다.

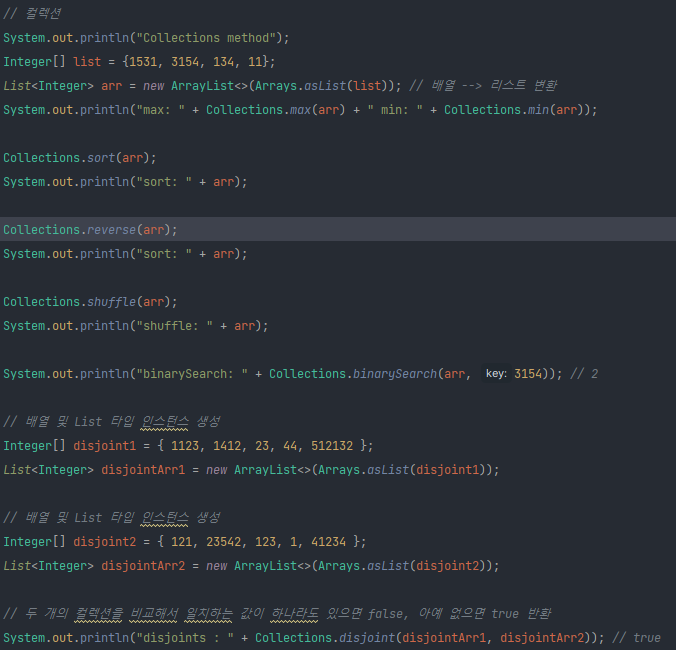


주요 인터페이스 간의 상속관계

1. 컬렉션 프레임워크의 이점
   1. List, Queue, Set, Map 등의 인터페이스를 제공하고 이를 구현하는 클래스를 제공하여 검증되고 최적화된 API를 사용할 수 있다.
   2. 가변적인 저장 공간을 제공한다. 고정적인 저장 공간을 제공하는 배열에 대비되는 특징이다.
   3. 자료구조, 알고리즘을 구현하기 위한 코드를 직접 작성할 필요 없이, 이미 구현된 컬렉션 클래스를 목적에 맞게 선택하여 사용하면 된다.

|  |  |
| --- | --- |
| 오류구분 | 설명 |
| Interfaces (인터페이스) | 컬렉션들이 가져야 하는 조직에 대한 설명과 함께 기능들을 추상적으로 표현한 것들이다. 예를 들자면 객체에 대한 검색과 삭제에 관련된 기능들에 대한 목록이다. 그리고 이것은 계층적인 구조를 이루게 한다. |
| Implementaionts (구현 객체) | 위의 인터페이스들을 구체적으로 구현한 클래스들을 의미한다. 그러므로 재사용 할 수 있도록 하는 자료의 구조인 것이다. |
| Algorithms (메서드) | 인터페이스를 구현한 객체들의 검색 그리고 정렬과 같은 유용한 동작들, 즉 메서드들을 의미한다. |

■ Collections 메소드



2. Set 인터페이스

: 특별한 기준에맞춰서 정렬되지 않지만 저장되는 객체들간의 중복된 요소가 발생하지 못하도록 내부적으로 관리되고 있다.

|  |  |
| --- | --- |
| 구현 클래스 | 설명 |
| HashSet Set | 인터페이스를 구현하고 있으며 내부적으로 HashMap을 사용하고 있다. 얻어지는 Iterator의 정렬 상태를 보장하지 못하므로 특정한 기준으로 정렬을 이루고 있지 않으며 저장 및 검출과 같은 동작에는 일정한 시간을 필요로 한다. |
| TreeSet | 내부적으로 Set 인터페이스를 구현하고 있으며 TreeMap에 의해 후원을 받는다. 그리고 기본적으로 얻어지는 Iterator의 요소들은 오름차순 정렬 상태를 유지하고 있다. |

3. List 인터페이스

: 시퀀스라고도 하며 시작과 끝이 선정되어 저장되는 요소들을 일괄적인 정렬 상태를 유지하면서 유소들의 저장이 이루어진다.

■ Stack : 후입선출(LIFO) 구조, push()로 요소를 빼내고 pop()으로 요소를 삽입한다.

■ Vector : 객체를 저장하는 배열과 같은 동작을 하지만 고정된 길이의 배열과 달리 필요할 때 용량이 자동으로 증가하는 가변적 길이의 특징을 가졌다.

■ ArrayList : 배열의 크기를 조작하기 위한 여러 메소드가 있다. 스레드 동기화 기능을 제공하지 않는 다는 것 외에는 모든 것이 Vector와 같은 클래스이기 때문이다.

4. Queue 인터페이스

: 선입선출(FIFO) 구조

■ LinkedList: add(), poll() 메서드에 의해 선입선출하는 스레드 동기화를 제공하지 않는 클래스이다.

5. Map 인터페이스

: Key와 Value를 매핑하는 객체다. 여기에 사용되는 Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매핑할 수 있다. 정렬의 기준이 없으며 이는 마치 각 Value에 열쇠고리를 달아서 큰 주머니에 넣어두고 오로지 key로 각 Value를 참조할 수 있도록 해둔 구조라 할 수 있다.

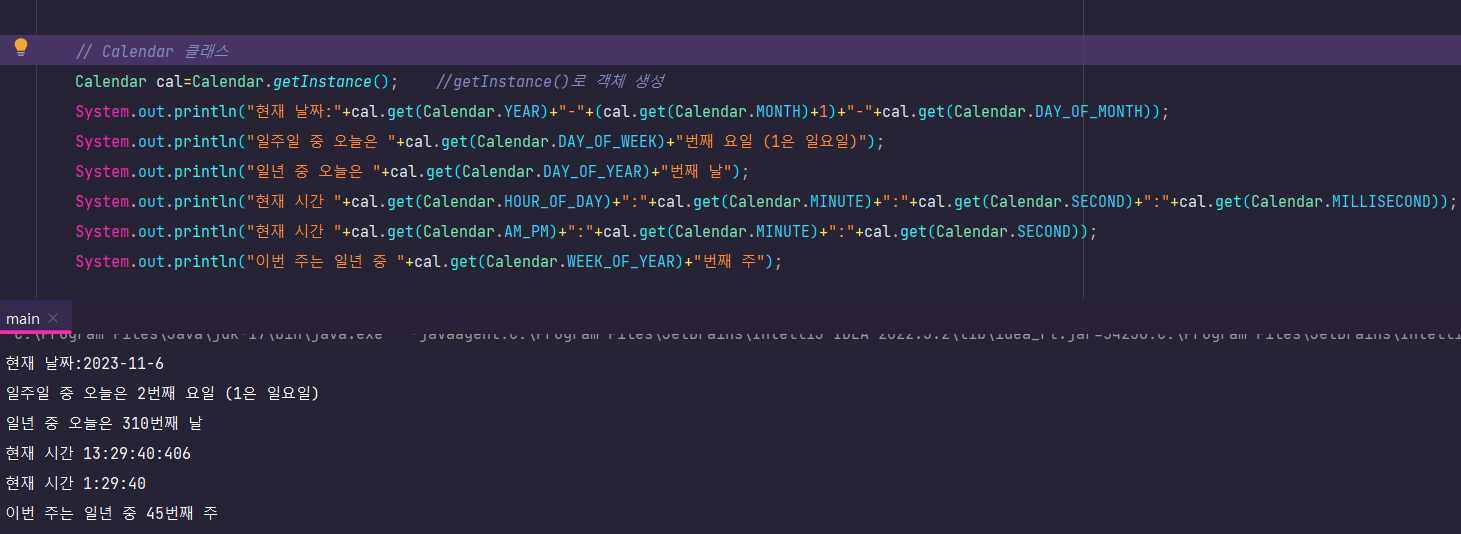
■ HashMap: Key와 Value를 하나의 쌍으로 저장되는 구조이며 저장되는 Value와 Key가 null을 허용한다. 하지만 중복을허용하지 않으므로 null을 가지는 Key가 두 개일 수 없다. 그리고 동기화가 포함되지 않았으므로 멀티스레드 환경에서의 구현이 아니라면 HashTable에 비해서 처리속도가 빠른 장점이 있다.

■ LinkedHashMap: HashMap은 순서대로 저장되지 않지만 이 클래스는 순서대로 저장된다.

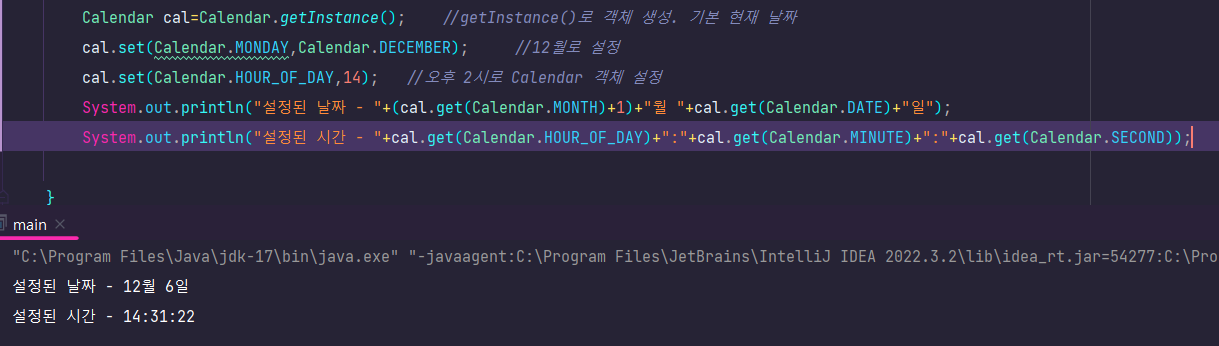
**SECTION 9. Util 패키지에 자주 쓰이는 클래스들**

1. Calendar 클래스

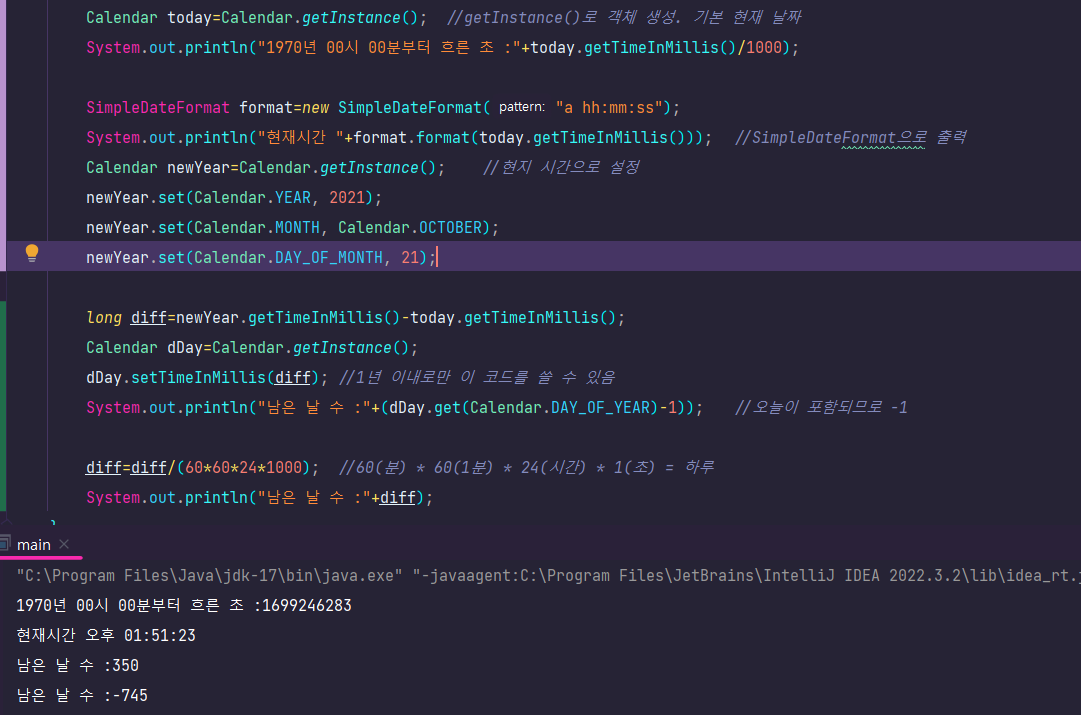
* 현재 시간의 정보를 표시



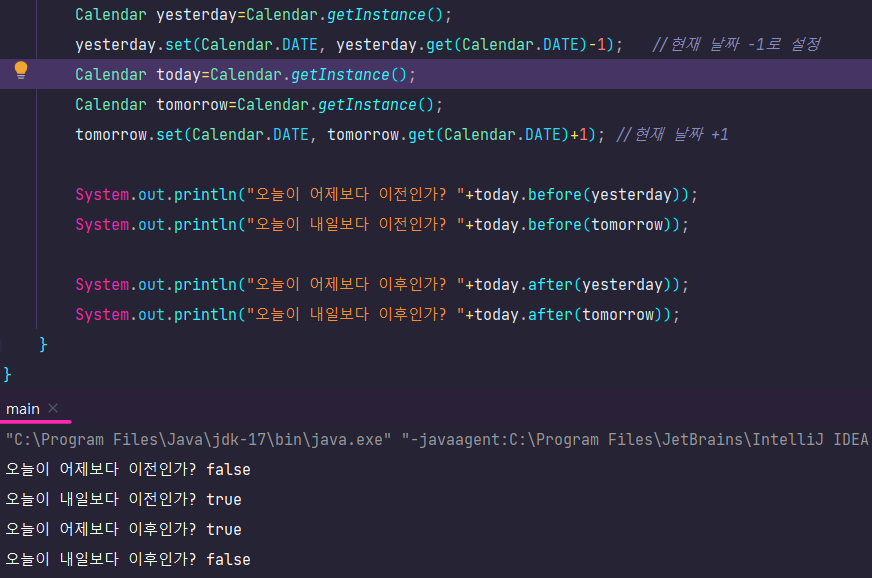
* Set 메소드로 날짜 설정



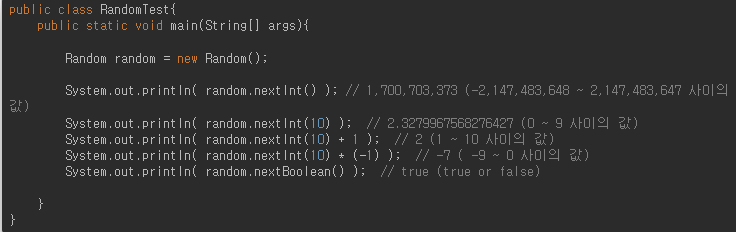
* 밀리초로 1970년 1월 1일 00시 00분부터 흐른 시간 구하기



* 특정 시간대가 현재 이후의 날인지, 이전의 날인지 확인



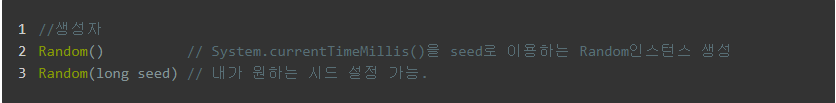
1. Random 클래스



* Math.random과 Random 클래스의 차이점

Math.random은 언제 어디서나 실행을 하면 항상 규칙성 없이 다른 수가 나온다. 하지만 Random 클래스는 일정한 규칙성을 가지고 있다. 또한 Random 클래스는 이 시드값을 설정해 줄 수 있기 때문에 랜덤 함수에서 같은 수가 반복되어 나오는 것을 방지할 수 있다. 그래서 seed값을 중간에 변경해 주어서 반복되는 것을 막고자 할 때 많이 사용된다.

Seed 값은 처음 생성자에 넣어서 만들어줄 수 있고,



코드의 흐름 중간에도 setSeed() 메소드를 사용해 seed 값을 변경해 줄 수 있다. 그리고 이 seed 값은 변경될 때마다 계속해서 변화가 이루어져야 함으로 현재시간을 천분의 1초 단위로 변환해서 반환해주는 System.currentTimeMillis()를 사용한다.

1. Arrays 클래스

: 배열을 다루기 위한 다양한 메소드가 포함되어 있다.

Arrays 클래스의 모든 메소드는 static method(클래스 메소드)이므로 객체를 생성하지 않고도 바로 사용할 수 있다.

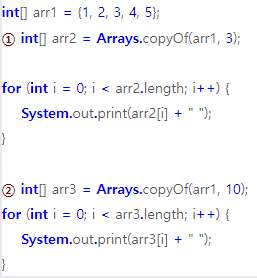
* BinarySearch 메소드

입력 배열에서의 특정 객체의 위치를 이진 검색 알고리즘을 사용하여 검색한 후 해당 인덱스를 반환한다.

* copyOf() 메소드

전달받은 배열의 특정 길이만큼을 새로운 배열을 복사하여 반환한다.

첫 번째 매새변수로 원본 배열을 전달받고 두 번째 매개변수로 원본 배열에서 새로운 배열로 복사할 요소의 개수를 전달받는다. 그리고 원본 배열과 같은 타입의 복사된 새로운 배열을 반환한다.



SECTION 10. AWT

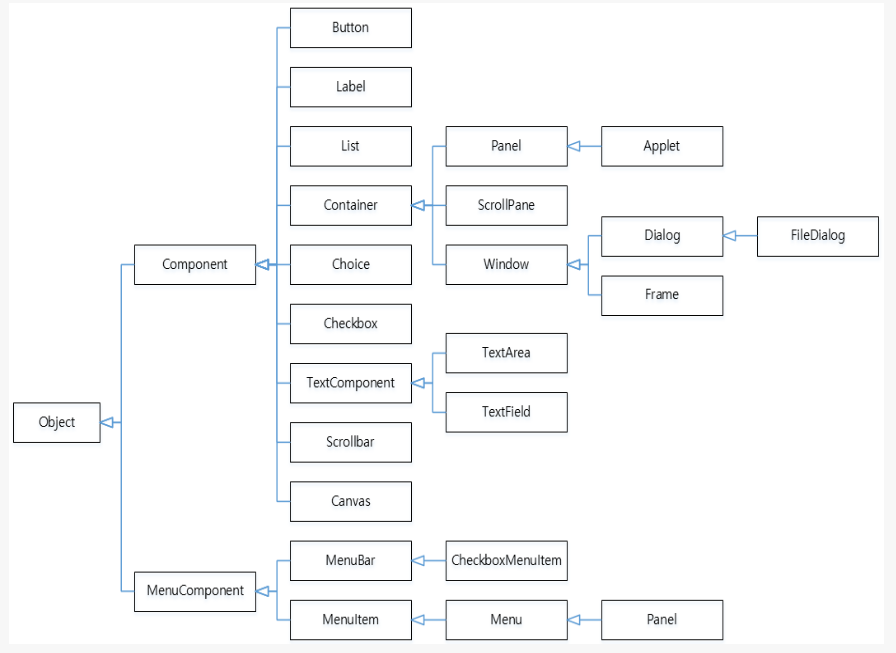
## AWT 기본 개념

: 윈도우 프로그래밍을 하기 위해 제공하는 라이브러리를 모아놓은 것이다.

: 모든 GUI 프로그래밍에 사용되는 컴포넌트와 툴킷을 제공하고 있으며 JFC와 같은 스윙 및 java2D의 모태가 되는 개념이다.

: java도 그렇듯 AWT는 운영체제에 구애받지 않고 쓸 수 있다. 하지만 운영체제의 리소스를 사용하기 때문에 AWT를 사용한 자바 GUI 프로그램은 실행되는 운영체제에 따라 다르게 보이거나 동작 방식에 차이가 있을 수 있다.

## Java.awt 패키지



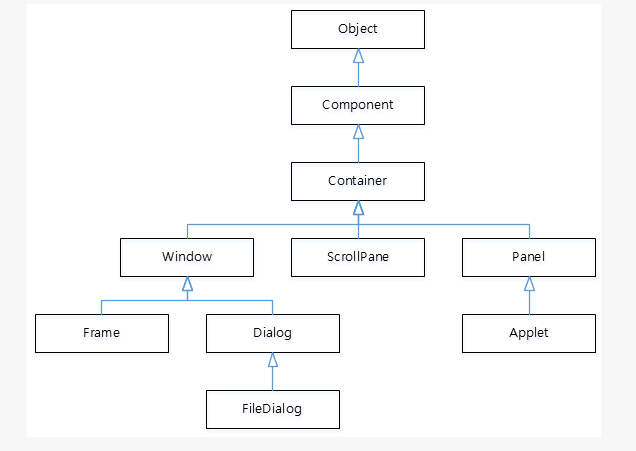
java.awt 패키지 구조도

## 컨테이너

: 컨테이너는 자신의 영역에 컴포넌트를 포함시키고 관리하는 역할을 하며 컨테이너가 다른 컨테이너를 포함할 수도 있다. 컨테이너는 주로 다른 컴포넌트를 포함시켜 사용하는 경우에 의미가 있다. 달리 표현하면 컨테이너에 컴포넌트를 부착시키지 않으면 독자적으로 화면에 출력할 수가 없고 반드시 컨테이너에 컴포넌트를 부착시켜야만 화면에 출력할 수 있다.

: 컨테이너의 종류에는 Frame, Window, Panel, Applet, Dialog, FileDialog, ScrollPane이 있고, 이러한 컨테이너에 컴포넌트를 부착시키기 위해 add() 메서드를 사용한다.

▶ 컨테이너의 상속 관계



## Frame 클래스

: Frame 클래스는 Window 클래스의 하위 클래스로 일반적인 응용 프로그램에서 윈도우를 생성하기 위해 사용되는 클래스다. Frame 클래스의 상위 클래스인 Window 클래스는 타이틀, 메뉴 등이 지원되지 않기 때문에 일반적으로 사용하지 않고 Frame 클래스를 사용한다.

: Frame 클래스는 기본 사이즈가 정해져 있지 않기 때문에 setSize(), setBounds() 메서드등을 이용해서 윈도우 크기를 설정한 후 setVisible(), show() 메서드를 통해서 화면에 출력시킬 수 있다.

▶ 주요 메서드

* getExtendedState(): 프레임의 상태를 얻어온다.
* getFrames(): 애플리케이션에서 생성한 모든 프레임을 리턴한다.
* getMenuBar(); 프레임의 메뉴바를 얻어온다.
* getState(): 프레임의 상태를 얻어온다.
* getTitle(): 프레임의 타이틀을 얻어온다.
* setTitle(String title): 프레임의 타이틀을 지정한다.
* setState(String state): 프레임의 상태를 지정한다.
* setMenubar(MenuBar bar): 프레임의 메뉴바를 지정한다.

## Panel 클래스

: 컴포넌트들을 그룹별로 묶어서 처리할 때 주로 사용한다. 일반적으로 Frame에 컴포넌트를 직접 붙이지 않고 Panel에 그룹별로 붙이고 다시 Panel을 Frame에 붙이는 경우가 많다. 또한 Panel은 다른 Panel을 생성하여 자신에게 붙일 수도 있어 윈도우 프로그램을 만들 때는 여러 개의 panel을 사용하는 경우가 많다.

## Dialog 클래스

: 메인 윈도우 외에 메시지를 출력하거나 사용자로부터 자료를 입력받을 때 주로 사용하는 컨테이너이다.

: 보통은 Dialog 클래스로부터 상속받아 새로운 기능을 가진 대화상자를 만드는 데 사용된다.

## 컴포넌트

: 타 컴포넌트에 독립적으로 설계되고 구현된 추상화와 캡슐화 단위로서, 다른 컴포넌트와 쉽게 결합되어 좀 더 큰 단위의 컴포넌트를 생성할 수 있으며, 응용 프로그램에서 다른 컴포넌트에 영향을 주지 않고 하나의 컴포넌트를 쉽게 교체할 수 있다. 즉, 개별적인 단위로 매매와 유통할 수 있으며 그 자체로서 재사용이 가능한 SW 단위이다.

▶ 컴포넌트의 크기 및 위치와 관련있는 메소드

* getX() : 컴포넌트의 현재 X 좌표를 얻어온다.
* getY(): 컴포넌트의 현재 Y 좌표를 얻어온다.
* getWidth(): 컴포넌트의 현재 폭을 얻어온다.
* getHeight(): 컴포넌트의 현재 높이를 얻어온다.
* getSize(): 컴포넌트의 크기를 크기 객체로 얻어온다.
* getMaximumSize(): 컴포넌트의 최대 크기를 크기 객체로 얻어온다.
* getMinimumSize(): 컴포넌트의 최소 크기를 크기 객체로 얻어온다.
* getBounds(): 컴포넌트의 경계를 직사각형 객체로 얻어온다.
* setSize(int width, int height): 컴포넌트의 폭, 높이를 지정한다.
* setLocation(int x, int y): 컴포넌트의 새로운 위치를 지정하여 이동시킨다.
* setBounds(int x, int y, int width, int height): 컴포넌트의 위치와 크기를 지정한다.
* setBounds(Rectangle r): 새로운 경계를 r에 적합하도록 컴포넌트의 위치와 크기를 지정한다.

▶ 컴포넌트의 색상, 폰트와 관련있는 메소드

* getBackground()
* getForeground()
* getFont()
* setFont(Font f)

▶ 컴포넌트의 설정과 관련있는 메소드

* setVisible()
* setEnable()
* getName()
* getParent(): return Container
* requestFocus(): 현 컴포넌트에 포커스를 요청한다.

#### 기본 컴포넌트 종류

|  |  |
| --- | --- |
| 종류 | 설명 |
| Button | 버튼 만들 때 사용 |
| Canvas | 비어 있는 공간으로 그래픽을 처리할 때 사용 |
| CheckBox | 체크 박스/라디오버튼 만들 때 사용 |
| Choice | Drop-down 리스트를 만들 때 사용 |
| Label | 고정된 문자열을 만들 때 사용 |
| List | 리스트를 만들 때 사용 |
| Scrollbar |  |

#### Button 클래스

: 버튼을 사용자가 눌렀을 때 특정한 액션을 취할 수 있도록 만든 컴포넌트

▶ 주요 메소드

* addActionListener(ActionListener a): 버튼으로부터 액션 이벤트를 받기 위해 지정된 액션 리스너를 추가한다.
* getActionCommand(): 버튼에서 발생되는 액션 이벤트와 커멘드명을 얻어온다.
* getLabel(): 버튼의 라벨을 얻어온다.
* setLabelString(String label): 버튼의 라벨을 지정한 label로 설정한다.

#### CheckBox 클래스

* addItemListener(itemListener i): 체크박스로부터 아이템 이벤트를 받기 위해 지정된 아이템 리스너를 추가한다.
* getLabel(): 체크박스의 라벨을 얻어온다.
* setLabel(String label): 체크박스의 라벨을 지정한다.
* getState(): 체크박스가 On 또는 Off 상태인지 얻어온다.
* setState(Boolean state)
* setCheckboxGroup(CheckboxGroup group)

#### Choice 클래스

: List 클래스와 거의 유사한 기능을 가지고 있는 컴포넌트로 사용자가 drop-down 버튼을 사용하여 여러 아이템 중에 하나를 선택할 수 있는 기능을 제공한다. 컴포넌트를 생성한 후 drop-down 리스트 항목에 추가시켜 사용한다.

▶주요 메소드

* add(String item): 메뉴에 항목 추가
* insert(String item, int idx)
* remove(int position)
* remove(String item)
* removeAll()
* getItemCount()
* getItem(int idx)
* getSelectedIndex()
* getSelectedItem()
* select(int idx): 지정한 위치의 항목을 선택한다.
* select(String str): 지정한 이름의 항목을 선택한다.

#### Label 클래스

▶ 주요 멤버 필드

* CENTER: return static int
* LEFT: return static int
* RIGHT: return static int

#### List 클래스

: 여러 개의 항목을 보여주고 사용자가 하나 또는 여러 개의 항목을 선택할 수 있도록 지원하는 컴포넌트. 즉 기본적으로 하나의 항목만을 선택할 수 있지만 MultipleMode를 설정하면 한 번에 여러 개의 항목을 선택할 수 있다.

▶ 주요 생성자

* List() : 새로운 리스트 객체 생성
* List(int rows): rows 만큼의 item을 갖는 리스트 객체 생성
* List(int rows, Boolean multipleMode) : rows만큼의 item을 갖는 리스트 객체 생성

#### Canvas 클래스

: 특정한 모양을 갖고 있지 않고 사각형의 영역만을 가지고 있는 컴포넌트

: Canvas 클래스는 그래픽이나 이미지를 표현하는 데 쓰이는 객체이다.

: 따로 Canvas 클래스를 상속 받는 클래스를 따로 작성하여 paint 메소드나 update 메소드를 오버라이딩 해야 한다.

▶ 주요 메서드

* Paint(Graphics g) : 캔버스에 그림 그릴 때 사용
* Update(Graphics g): 캔버스를 업데이트 할 때 사용

### 텍스트 컴포넌트

: java.awt.TextComponent 클래스는 텍스트를 다루는 클래스의 super 클래스로 텍스트를 처리하는 각종 메서드를 가지고 있다. 그렇지만 독립적으로는 생성되지 못한다,

#### TextField 클래스

: 한 줄 내의 텍스트를 입력받거나 편집할 수 있는 컴포넌트

: 한 줄에 표시할 수 있는 컬럼수를 지정할 수 있고 반향 문자를 지정하면 입력되는 문자 대신 반향 문자로 지정한문자로 출력된다.

#### TestArea 클래스

: 여러 줄의 텍스트를 사용자로부터 입력받거나 편집할 수 있는 컴포넌트다.

화면에 출력되는 영역이 벗어나면 스크롤바 표시 방식에 따라 자동으로 스크롤바가 생성된다.

### 메뉴 컴포넌트

: 메뉴는 보통 최상위 레벨의 윈도우 타이틀바 아래에 존재하는 것으로 사용자가 프로그램의 기능을 선택할 수 있도록 해주는 기능을 가지고 있는 컴포넌트로 메뉴의 구성은 MenuBar, Menu, MenuItem으로 구성된다.

## 배치 관리자

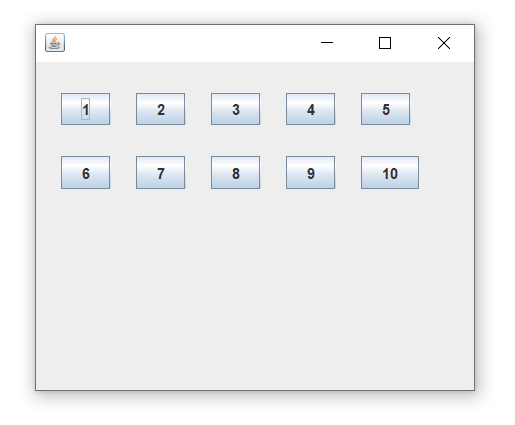
: 미리 정해진 레이아웃에 따라 컴포넌트들을 자동으로 배치하는 기능을 가지고 있는 객체를 가지고 있는데 이를 ‘배치 관리자’라고 한다.

: 배치 관리자에는 FlowLayout, BorderLayout, GridLayout, GridBagLayout, CardLayout이 있다.

|  |  |
| --- | --- |
| 컨테이너 | 기본 배치 관리자 |
| Frame | BorderLayout |
| Panel | FlowLayout |
| Dialog | BorderLayout |
| Applet | FlowLayout |

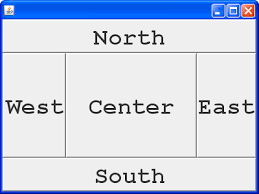
### FlowLayout

: 컴포넌트들을 수평으로 순서대로 늘어놓는 배치 기능을 가지고 있다. 처음에 배치를 하게 되면 상단, 중앙부터 배치되는데 배치를 하다가 더 이상 배치할 공간이 없으면 자동으로 다음 줄로 이동하여 배치하게 된다. 컴포넌트를 배치할 때 컴포넌트의 간격을 갭이라고 하는데 컴포넌트들 사이의 수평, 수직간의 간격을 설정할 수 있다.



### BorderLayout

: 컨테이너 영역을 아래와 같이 컨테이너의 영역을 5개 영역으로 분할하여 컴포넌트를 배치하는 관리자이다.



: CENTER 영역은 다른 영역에 아무것도 존재하지 않으면 그 영역까지 포함해서 영역이 잡힌다.

: SOUTH, NORTH 영역은 컴포넌트의 높이는 제대로 나타나지만 폭의 길이는 인정되지 않는다.

: WEST, EAST 영역은 컴포넌트 폭의 길이는 제대로 나타나지만 높이는 제대로 인정되지 않고 항상 그 영역의 길이만큼 잡히게 된다.

### GridLayout

: 격자모양과 같이 가로와 세로가 같은 크기의 비율로 나누어 각 공간에 컴포넌트를 배치할 수 있는 관리자다.

: GridLayout 배치 관리자를 만들 때 행과 열의 수를 지정하는데, 값을 0 이상의 값으로 지정하며 만약 0으로 지정하게 되면 무한대로 컴포넌트를 추가하여 붙일 수 있다. 또한 행과 열의 수하고 붙이는 컴포넌트의 수가 더 많은 경우는 행의 수를 우선으로 맞춘다. 만약 2행 3열짜리의 GridLayout을 만들었는데 컴포넌트 8개를 붙였다면 실제로 GridLayout의 크기는 2행 4열의 배열이 된다.

### GridBagLayout

: GridLayout은 하나의 셀에는 하나의 컴포넌트를 가질 수 있지만 이 관리자는 여러 셀에 걸쳐서 서로 다른 크기와 간격으로 하나의 컴포넌트가 배치될 수 있다. 그래서 GridBagLayout을 사용하는 경우는 GridBagLayout 클래스를 사용하여 배치시킨다. GridBagLayout 클래스는 GridLayout으로 지정된 컨테이너에 컴포넌트가 얼마만큼의 영역을 차지하여 배치할 것인가에 대한 자세한 영역 구조에 대해 지정한다.

SECTION 11. AWT 이벤트

프로그램이 실행되면서 운영체제가 해당 프로그램에서 이벤트가 발생하는지를 검사한다. 이때 이벤트가 발생하면 운영체제가 JVM에게 이벤트를 전달하고 JVM은 발생된 이벤트를 처리하기 위하여 이벤트 객체를 생성한다. 그런 후 이벤트를 처리하기 위해 이벤트 객체를 가지고 핸들러를 호출한다.

* 이벤트 소스

: 이벤트가 발생되는 컴포넌트를 말한다. 즉, 버튼, 체크박스, 리스트, 프레임, 마우스 등과 같은 컴포넌트들이 이벤트 소스이다.

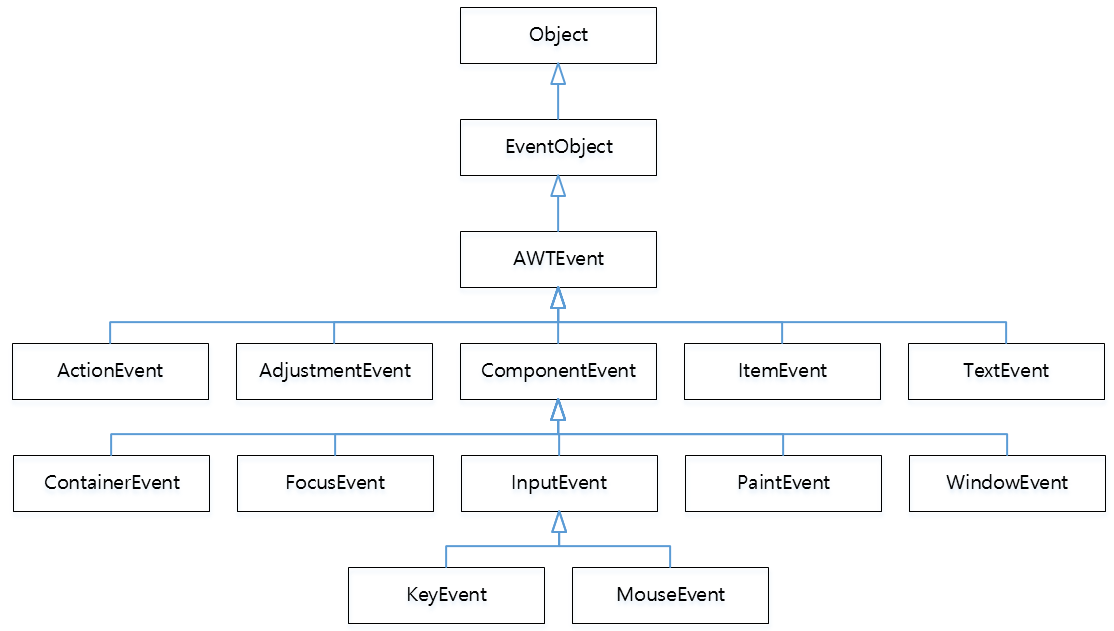
* 이벤트 리스너

: 이벤트 소스에서 이벤트가 발생하는지를 검사하고 있다가 이벤트가 발생되면 실제적으로 이벤트를 처리할 수 있도록 만든 인터페이스이다.

* 이벤트 핸들러

: 이벤트 리스너에 전달된 이벤트를 실제로 처리할 수 있도록 이벤트 리스너에 포함되어 있는 메서드로 발생된 이벤트 객체를 받아와서 실제적으로 처리해주는 기능이 있다.

## 이벤트 클래스 구조와 종류



여러 이벤트 종류에 대한 세부적인 사항들은 Java DOC을 참고하자.

### ItemEvent

: 체크박스, 리스트, Choice 컴포넌트에서 항목이 선택되거나 선택이 해제되었을 때 발생하는 이벤트

### TextEvent

: 텍스트 컴포넌트(Text Field, TextArea)에서 키가 입력되어 내용이 바뀌었을 때 발생하는 이벤트다. 이 이벤트는 내용이 바뀔 때마다 발생하므로 주의해서 사용해야 된다. 사용자가 입력할 때마다 처리되는 경우가 있을 경우에도 사용하는 이벤트이다.

### KeyEvent

: 사용자가 키보드와 같은 입력장치를 통해서 키 입력을 했을 때 발생하는 이벤트다.

### WindowEvent

: 윈도우가 활성화, 비활성화, 아이콘화 및 창이 닫힐 때 발생하는 이벤트다.

## 리스너 종류

: 리스너는 각각의 Event를 처리하는 이벤트 리스너이다.

: ActionListener, ItemListener, TextListener, KeyListener, MouseListener, MouseMotionListener, WindowListener가 있다.

## 이벤트 처리

### 컴포넌트 이벤트 처리 3단계

: 자바에서 이벤트를 처리해주는 클래스는 이미 정의되어 있다. 그러므로 프로그래밍할 때엔 이벤트 소스와 리스너에 중점을 두고 프로그래밍하면 된다.

* 1단계: 이벤트 소스 결정

하나의 윈도우에는 여러 개의 컴포넌트가 존재할 수 있으므로 실제로 이벤트가 발생되면 처리할 컴포넌트를 결정한다.

* 2단계: 이벤트 리스너 작성

이벤트를 실제적으로 처리할 수 있도록 해당 이벤트를 처리할 이벤트 리스너 인터페이스를 이용해서 이벤트 리스너 클래스를 작성한다.

* 3단계: 이벤트 소스와 이벤트 리스너 연결

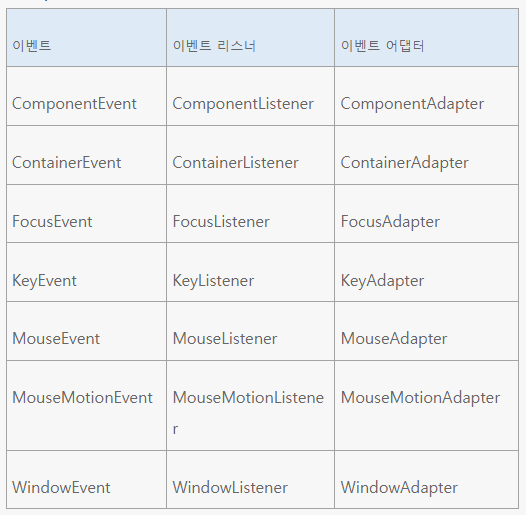
이벤트 리스너가 작성되면 리스너와 이벤트 소스와 연결하여 이벤트 소스에서 실제적으로 이벤트가 발생되면 처리할 수 있도록 addXXXXListener() 함수를 통해 연결을 시켜준다. XXXX 부분은 해당 컴포넌트에 붙일 수 있는 리스너 이름이다.

## 이벤트 Adapter 클래스

Adapter 클래스를 통해서 인터페이스로 되어있는 이벤트 리스너의 추상 메소드를 모두 오버라이딩 시켜야 사용하는 불편함을 덜어준다.

Adapter 클래스는 이벤트 리스너 인터페이스 중에서 추상메소드가 두 개 이상 존재하는 인터페이스를 구현한 추상 클래스이며 인터페이스에 있는 모든 메소드를 빈 메소드로 재정의했기 때문에 인터페이스를 구현하여 불필요한 메소드를 재정의하는 수고를 덜어준다. 즉, Adapter 클래스를 상속받은 클래스에서는 자신이 필요한 메소드만을 재정의할 수 있다.

### Adapter클래스의 종류



### Adapter 클래스의 활용

: Adapter 클래스로부터 상속받는 클래스를 생성하여 처리하는 것보다 익명 클래스를 이용하는 방법과 내부 클래스를 이용하는 방법이 있다.