1.API(Application programming Interface)

> API 혹은 라이브러리란 개발에 자주 사용되는 클래스나 인터페이스의 모음이다.
>
> System,String 클래스 등도 모두 API 에 속한다.

2.lang, util 패키지

2-1. java.lang 패키지

자바 프로그램의 기본적 클래스를 담고 있다.(Object,Class,System,String, ...)

import 필요 없다.

2-2. java.util

기본적 패키지2

Collection Class,Arrays,Calender,Date,StringTokenizer,Random 등

3. Object 클래스

---

java.lang.Object 클래스

java.lang 패키지 중에서도 가장 많이 사용되는 클래스는 바로 Object 클래스입니다.

Object 클래스는 모든 자바 클래스의 최고 조상 클래스가 됩니다.

따라서 자바의 모든 클래스는 Object 클래스의 모든 메소드를 바로 사용할 수 있습니다.

이러한 Object 클래스는 필드를 가지지 않으며, 총 11개의 메소드만으로 구성되어 있습니다.

Object 메소드

Object 클래스의 메소드는 다음과 같습니다.

메소드	설명
protected Object clone()	해당 객체의 복제본을 생성하여 반환함.

메소드	설명
boolean equals(Object obj)	해당 객체와 전달받은 객체가 같은지 여부를 반환함.
protected void finalize()	해당 객체를 더는 아무도 참조하지 않아 **가비지 컬렉터가 객체의 리소스를 정리하기 직전에 호출** **하지만 자동으로 관리시키면 GC의 무작위성 순서 때문에 예측이 힘드므로 명시적으로 자원해제 해주는게 좋다.**
Class<T> getClass()	해당 객체의 클래스 타입을 반환함.
int hashCode()	해당 객체의 해시 코드값을 반환함.
void notify()	해당 객체의 대기(wait)하고 있는 하나의 스레드를 다시 실행할 때 호출함.
void notifyAll()	해당 객체의 대기(wait)하고 있는 모든 스레드를 다시 실행할 때 호출함.
String toString()	해당 객체의 정보를 문자열로 반환함.
void wait()	해당 객체의 다른 스레드가 notify()나 notifyAll() 메소드를 실행할 때까지 현재 스레드를 일시적으로 대기(wait)시킬 때 호출함.
void wait(long timeout)	해당 객체의 다른 스레드가 notify()나 notifyAll() 메소드를 실행하거나 전달받은 시간이 지날 때까지 현재 스레드를 일시적으로 대기(wait)시킬 때 호출함.
void wait(long timeout, int nanos)	해당 객체의 다른 스레드가 notify()나 notifyAll() 메소드를 실행하거나 전달받은 시간이 지나거나 다른 스레드가 현재 스레드를 인터럽트(interrupt) 할 때까지 현재 스레드를 일시적으로 대기(wait)시킬 때 호출함.

---

toString() 메소드

toString() 메소드는 해당 인스턴스에 대한 정보를 문자열로 반환합니다.

이때 반환되는 문자열은 클래스 이름과 함께 구분자로 '@'가 사용되며, 그 뒤로 16진수 해시 코드(hash code)가 추가됩니다.

16진수 해시 코드 값은 인스턴스의 주소를 가리키는 값으로, 인스턴스마다 모두 다르게 반환됩니다.

다음 예제는 toString() 메소드를 이용하여 인스턴스의 정보를 출력하는 예제입니다.

예제

```
Car car01 = new Car();

Car car02 = new Car();


System.out.println(car01.toString());
```

```
System.out.println(car02.toString());
```

코딩연습 ▶

실행 결과

Car@15db9742

Car@6d06d69c

> 자바에서 toString() 메소드는 기본적으로 각 API 클래스마다 자체적으로 오버라이딩을 통해 재정의되어 있습니다.

---

equals() 메소드

equals() 메소드는 해당 인스턴스를 매개변수로 전달받는 참조 변수와 비교하여, 그 결과를 반환합니다.

이때 참조 변수가 가리키는 값을 비교하므로, 서로 다른 두 객체는 언제나 false를 반환하게 됩니다.

다음 예제는 equals() 메소드를 이용하여 두 인스턴스를 서로 비교하는 예제입니다.

예제

```
Car car01 = new Car();

Car car02 = new Car();


System.out.println(car01.equals(car02));

car01 = car02; // 두 참조 변수가 같은 주소를 가리킴.

System.out.println(car01.equals(car02));
```

코딩연습 ▶

실행 결과

false

true

자바에서 equals() 메소드는 기본적으로 각 API 클래스마다 자체적으로 오버라이딩을 통해 재정의되어 있습니다.

---

clone() 메소드

clone() 메소드는 해당 인스턴스를 복제하여, 새로운 인스턴스를 생성해 반환합니다.

하지만 Object 클래스의 clone() 메소드는 단지 필드의 값만을 복사하므로, 필드의 값이 배열이나 인스턴스면 제대로 복제할 수 없습니다.(*배열의 주소만 복사하는 **얕은복사 주의***)

따라서 이러한 경우에는 해당 클래스에서 clone() 메소드를 오버라이딩하여, 복제가 제대로 이루어지도록 재정의해야 합니다.

이러한 clone() 메소드는 데이터의 보호를 이유로 Cloneable 인터페이스를 구현한 클래스의 인스턴스만이 사용할 수 있습니다.

다음 예제는 clone() 메소드를 이용하여 인스턴스를 복제하는 예제입니다.

예제

```
import java.util.*;


class Car implements Cloneable {
    private String modelName;
① private ArrayList<String> owners = new ArrayList<String>();

    public String getModelName() { return this.modelName; }                  // modelName의 값을 반환함
    public void setModelName(String modelName) { this.modelName = modelName; } // modelName의 값을 설정함


    public ArrayList getOwners() { return this.owners; }                     // owners의 값을 반환함
    public void setOwners(String ownerName) { this.owners.add(ownerName); }   // owners의 값을 추가함
```

```
    public Object clone() {
        try {
②           Car clonedCar = (Car)super.clone();
③           // clonedCar.owners = (ArrayList)owners.clone();
            return clonedCar;
④       } catch (CloneNotSupportedException ex) {
            ex.printStackTrace();
            return null;
        }
    }
}

public class Object03 {
    public static void main(String[] args) {
⑤       Car car01 = new Car();
        car01.setModelName("아반떼");
        car01.setOwners("홍길동");
⑥       System.out.println("Car01 : " + car01.getModelName() + ", " + car01.getOwners() + "₩n");

⑦       Car car02 = (Car)car01.clone();
⑧       car02.setOwners("이순신");
⑨       System.out.println("Car01 : " + car01.getModelName() + ", " + car01.getOwners());
⑩       System.out.println("Car02 : " + car02.getModelName() + ", " + car02.getOwners());
    }
}
```

코딩연습 ▶

실행 결과

Car01 : 아반떼, [홍길동]

Car02 : 아반떼, [홍길동, 이순신]

Car02 : 아반떼, [홍길동, 이순신]

위 예제의 ②번 라인에서는 부모 클래스의 clone() 메소드를 호출하여 clone() 메소드를 오버라이딩하고 있습니다.

⑤번 라인에서는 Car 클래스의 인스턴스인 car01을 생성하고, ⑦번 라인에서는 오버라이딩한 clone() 메소드를 호출하여 복제를 수행하고 있습니다.

하지만 ②번 라인처럼 clone() 메소드를 재정의하면, 필드의 값이 ①번 라인처럼 인스턴스일 때는 제대로 된 복제를 수행할 수 없습니다.

⑧번 라인에서는 복제된 인스턴스인 car02의 owners 필드에 새로운 값을 하나 추가합니다.

하지만 ⑨번 라인의 실행 결과를 보면, ⑦번 라인의 결과와는 달리 원본 인스턴스인 car01의 owners 필드에도 새로운 값이 추가되었음을 확인할 수 있습니다.

이처럼 단순히 부모 클래스의 clone() 메소드를 호출하여 clone() 메소드를 재정의하면, 배열이나 인스턴스인 필드는 복제되는 것이 아닌 해당 배열이나 인스턴스를 가리키는 주소값만이 복제되는 것입니다.

따라서 정확한 복제를 위해서는 ③번 라인처럼 배열이나 인스턴스인 필드에 대해서는 별도로 clone() 메소드를 구현하여 호출해야 합니다.

③번 라인의 주석을 해제하고 결과보기를 다시 실행하면, 다음과 같이 정확한 실행 결과가 출력될 것입니다.

실행 결과

Car01 : 아반떼, [홍길동]

Car02 : 아반떼, [홍길동]

Car02 : 아반떼, [홍길동, 이순신]