6주차 과제

study 할래?

스터디 내용

목표 : 자바의 상속에 대해 학습

1. 자바 상속의 특징
2. super 키워드
3. 메소드 오버라이딩
4. 다이나믹 메소드 디스패치 (Dynamic Method Dispatch)
5. 추상 클래스
6. final 키워드
7. Object 클래스

1. Java 상속의 특징

상속이란?

상속이라는 단어의 뜻과 마찬가지로 자바에서도 상속은 부모클래스의 변수와 메소드를 물려받는것을 말한다. 이런 상속은 코드의 재사용성을 통해 코드의 간결성을 확보해준다.

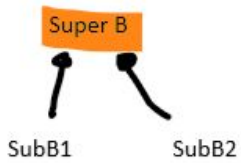
특징

1. 상속은 단일 상속만 가능하다.
2. 자바의 계층 구조 최상위에는 java.lang.Object 클래스가 존재한다.
3. 자바에서는 상속의 횟수에 제한을 두지 않는다.
4. 부모의 메소드와 변수만 상속되며, 생성자는 상속되지 않는다.
   (부모의 메소드는 재정의 하여 사용 가능하다 - 오버라이딩)

1. 상속 메모리

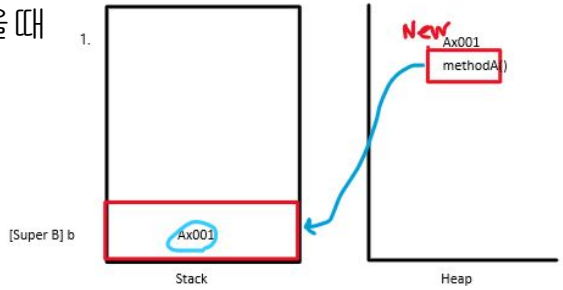
"그렇다면 오버라이딩을 했을때 슈퍼클래스의 메서드를 사용하고 싶다면?"

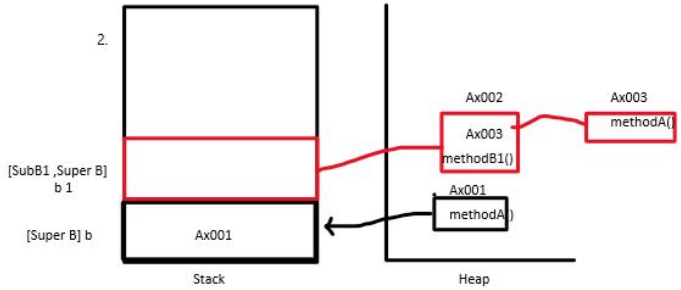
```
1  class SuperB{
2  public void methodA(){
3      System.out.println("SuperB's methodA()");
4  }
5  }//SuperB
6  class SubB1 extends SuperB{
7      public void methodB1(){
8          System.out.println("SubB1's methodB1()");
9      }
10 }
11 class SubB2 extends SuperB{
12     public void methodA(){
13         System.out.println("SubB2's methodA()");
14     }
15 }
16
17 public class InherTest_02 {
18     public static void main(String[] args) {
19         SuperB b=new SuperB();
20         b.methodA();
21         SubB1 b1=new SubB1();
22         b1.methodA();
23         SubB2 b2=new SubB2();
24         b2.methodA();
25     }//main
26 }
27
```

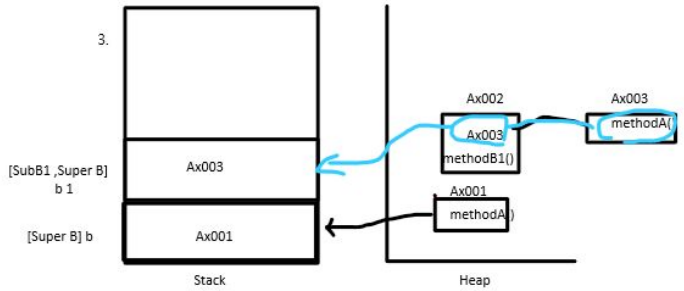



```
SuperB's methodA()
SuperB's methodA()
SubB2's methodA()
```

Stack 메모리는 컴파일할때 잡힘







2. SUPER 키워드

"그렇다면 오버라이딩을 했을때 슈퍼클래스의 메서드를 사용하고 싶다면?"

→ SUPER 키워드를 사용하면 된다.

Super 키워드는 자식클래스가 부모클래스로부터 상속받은 멤버를 사용하고자 할때 사용된다.

```
SubB2's methodA()
SuperB's methodA()
```

```java
class SuperB{
    public void methodA(){
        System.out.println("SuperB's methodA()");
    }
}
class SubB1 extends SuperB {
    public void methodB1(){
        System.out.println("SubB1's methodB1()");
    }
}
class SubB2 extends SuperB {
    public void methodA(){
        System.out.println("SubB2's methodA()");
        super.methodA();
    }
}


public class InherTest_02 {
    public static void main(String[] args) {
        SubB2 b2 = new SubB2();
        b2.methodA();

    }
}
```

3. 메소드 오버라이딩

오버라이딩은 부모의 함수를 재정의하는 기능이다. 같은 동작에서 다른 기능을 구현해야 하는 경우가 있는데 그때 사용한다. 그렇기 때문에 함수명, 리턴값, 파라미터가 모두 동일해야한다.

```
class SuperB{
    public void methodA(){
        System.out.println("SuperB's methodA()");
    }
}
class SubB1 extends SuperB {
    public void methodB1(){
        System.out.println("SubB1's methodB1()");
    }
}
class SubB2 extends SuperB {
    public void methodA(){
        System.out.println("SubB2's methodA()");
        super.methodA();
    }
}


public class InherTest_02 {
    public static void main(String[] args) {
        SubB2 b2 = new SubB2();
        b2.methodA();

    }
}
```

4. 다이나믹 메소드 디스패치_메소드 디스패치

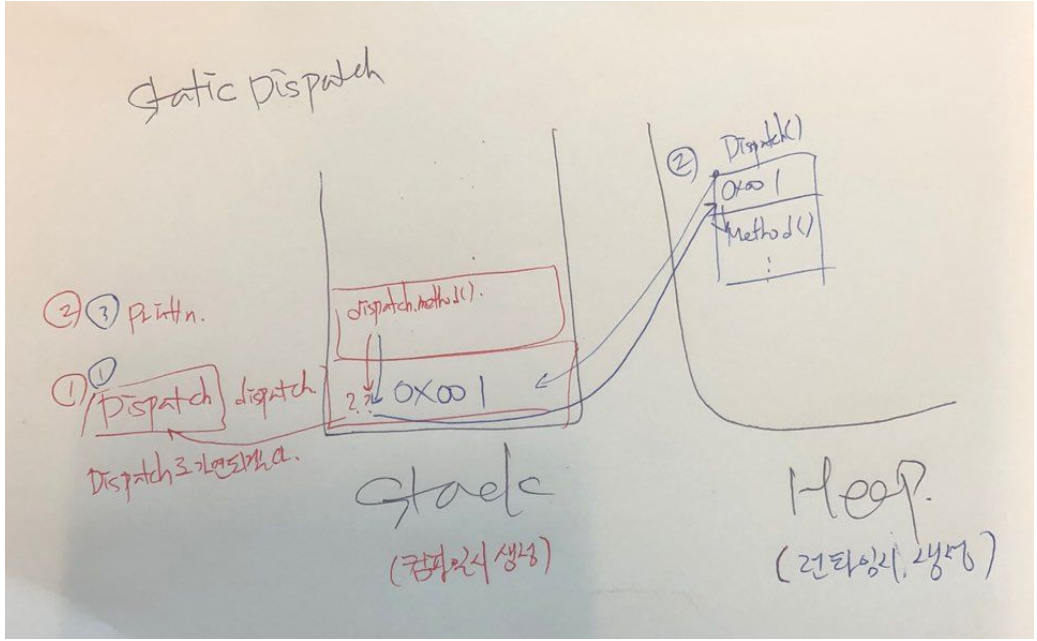
Dispatch 1. (특히 특별한 목적을 위해) 보내다 2. (편지,소포,메세지를) 보내다.
네이버 검색결과이다. 자바는 객체지향프로그래밍언어로서 객체들간의 메세지 전송을 기반으로 문제를 해결하게된다. 메세지 전송이라는 표현은 결국 메서드를 호출하는 것인데 그것을 dispatch라고 부르는 것이다.

1. Static Method Dispatch
   static은 구현클래스를 이용해 컴파일타임에서부터 어떤 메서드가 호출될지 정해져있다. 타입자체가 Dispatch라는 구현클래스이기때문에 해당 메서드를 호출하면 어떤 메서드가 호출될지 정적으로 정해진다. 이 정보는 컴파일이 종료된 후 바이트코드에도 드러나게 된다.

4.Static Dispatch

```
lic class DispatchTest {
    public static void main(String[] arg) {
        Dispatch dispatch = new Dispatch();
        System.out.println(dispatch.method());
    }
}

class Dispatch{
    public String method(){
        return "hello dispatch";
    }
}
```



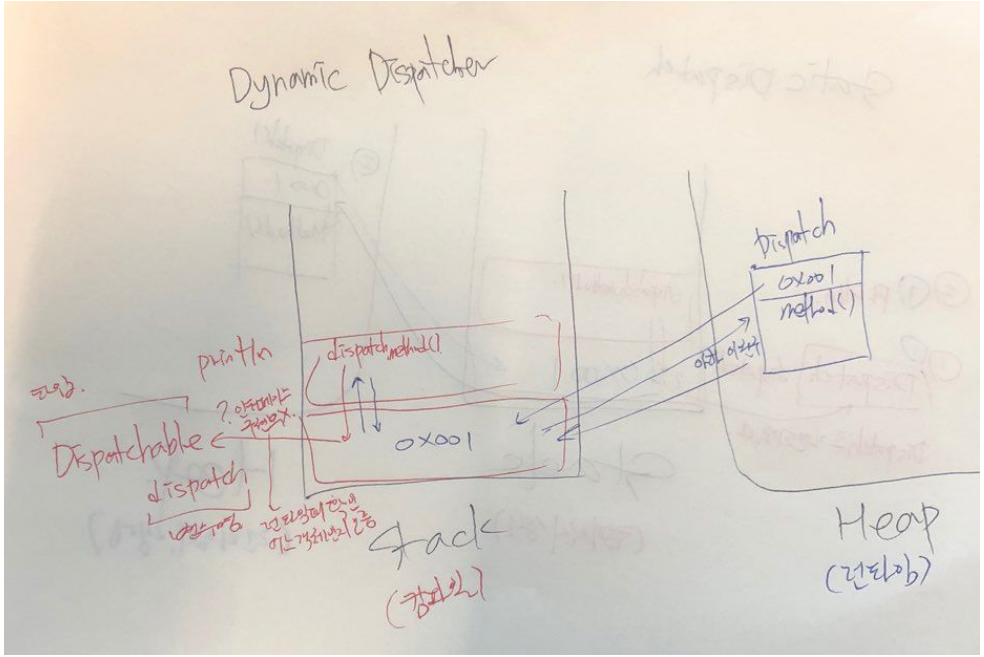
4. 다이나믹 메소드 디스패치_메소드 디스패치

2. Dynamic Method Dispatch

인터페이스 타입으로 메서드를 호출한다. 컴파일러는 타입에 대한 정보를 알고있으므로 런타임시에 호출 객체를 확인해 해당 객체의 메서드를 호출한다. 런타임시에 호출 객체를 알 수 있으므로 바이트코드에도 어떤 객체의 메서드를 호출해야하는지 드러나지 않는다.예제코드에서 method() 메서드는 인자가 없지만 자바는 묵시적으로 항상 호출 객체를 인자로 보내게된다. 호출 객체를 인자로 보내기때문에 this를 이용해 메서드 내부에서 호출객체를 참조할 수 있는 것이다.

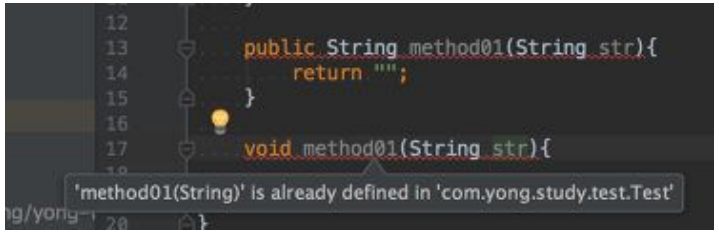
4. Dynamic Dispatch

```
2
3  public class DynamicDispatchTest {
4          public static void main(String[] arg) {
5              Dispatchable dispatch = new Dispatch();
6              System.out.println(dispatch.method());
7          }
8      }
9      class Dispatch implements Dispatchable {
10         public String method(){
11             return "hello dispatch";
12         }
13     }
14     interface Dispatchable{
15         String method();
16     }
17
```



4. Method Signature

메서드 시그니처는 그것만으로 메서드를 구분지을수 있는 근거가 되어야한다.
그에 따라 자바에서 메서드 시그니처는 메서드명(인자)가 된다. 유의할점은 반환 타입은 시그니처에 포함되지 않는다는 것이다. 결국 오버로딩이 되면 시그니처가 다르다고 보면 된다.



아래 메서드는 이미 존재하는 메서드라는 오류가 난다.
인자 타입이 달라지면 에러는 사라진다. 혹은 인자 타입이 같아도 개수가 달라지면 에러가 사라진다. 인자의 타입과 개수까지 시그니처에 포함된다.

```
public String method01(String str){
    return "";
}

void method01(int num){

}
```

```
public String method01(String str){
    return "";
}

void method01(String str1, String str2){

}
```

4. 더블디스패치

더블 디스패치는 Dynamic Dispatch를 두 번 하는 것을 의미한다.

윗 설명에서 디스패치가 무엇인지 알아보았다. 동적 디스패치는 메서드를 호출하는 인자가 구현체가 없는 인터페이스 타입인 경우 런타임시에 객체를 찾아 알맞는 메서드를 호출하는 것이다. 동적 디스패치를 이용한 코드이다.

```java
public class DoubleDispatchTest {
    public static void main(String[] arg) {
        List<SmartPhone> phoneList = Arrays.asList(new Iphone(), new Gallaxy());

        Game game = new Game();
        //phoneList.forEach(game::play); //1
        for(SmartPhone e : phoneList){ //2
            game.play(e);
        }
    }
}

interface SmartPhone{
}

class Iphone implements SmartPhone{

}

class Gallaxy implements SmartPhone{

}

class Game {
    public void play(SmartPhone phone) {
        System.out.println("game play [" +phone.getClass().getSimpleName()+ "]");
    }
}
```

스마트폰 list

Game 클래스 사용

스마트폰list에서 데이터를 하나씩 꺼내 e에 담고 game클래스의 play메서드로 보냄

```
[com.oopsw.test.Iphone@2a139a55, com.oopsw.test.Gallaxy@15db9742]
game play [Iphone]
game play [Gallaxy]
```

매개인자가 interface

스마트폰 리스트를 돌며 각각 게임을 play하고 있다. 스마트폰 리스트 인터페이스를 타입 파라미터로 전달했기 때문에 동적 디스패치로 인해 출력내용은 모두 다르다. 이는 인터페이스로 참조하고있는 객체 레퍼런스를 동적으로 추적하기 때문이다.

그럼 여기서

GAME PLAY가 구현체 별로 다르게 구현되어야 한다면 어떻게 될까?

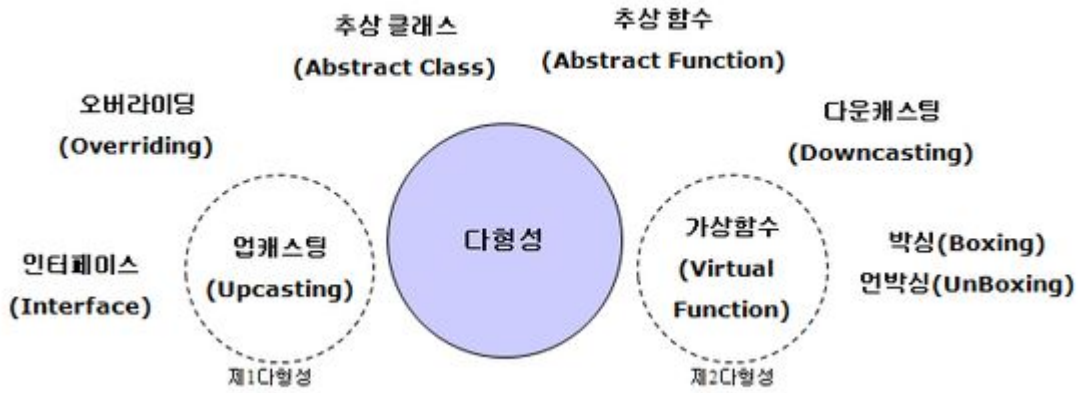
```java
public class DoubleDispatchTest {
    public static void main(String[] arg) {
        List<SmartPhone> phoneList = Arrays.asList(new Iphone(), new Gallaxy());
        Game game = new Game();
        //phoneList.forEach(game::play); //1
        for(SmartPhone e : phoneList){ //2
            game.play(e);
        }
    }
}
interface SmartPhone{
    public void game(Game game);
}
class Iphone implements SmartPhone{
    @Override
    public void game(Game game) {
        System.out.println("아이폰에서 실행합니다.");
    }
}
class Gallaxy implements SmartPhone{
    @Override
    public void game(Game game) {
        System.out.println("갤럭시에서 실행합니다.");
    }
}
class Game {
    public void play(SmartPhone phone) {
        phone.game(this);
        System.out.println("this:" + this);
        System.out.println("phone:" + phone);
    }
}
```

```
아이폰에서 실행합니다.
this:com.oopsw.test.Game@2a139a55
phone:com.oopsw.test.Iphone@15db9742
갤럭시에서 실행합니다.
this:com.oopsw.test.Game@2a139a55
phone:com.oopsw.test.Gallaxy@6d06d69c
```

메모리의 이동이
어떻게되는지?

5. 추상클래스

추상클래스는 클래스를 만들기 위한 일종의 설계도로 인스턴스를 생성할 수는 없는 클래스이다. 이를 사용하기 위해서는 반드시 자식 클래스에서 상속받아 클래스를 모두 구현해야만 한다. 동시에 추상클래스는 보통 하나 이상의 추상 메서드를 포함하고 있으며, 생성자와 멤버변수, 일반메서드 모두를 가질 수 있다.



5. 추상클래스 - abstract

```java
package com.oopsw.test;

abstract class Abstract{
    private int index = 9; //멤버변수를 가질 수 있다.
    public abstract void f1(); //추상클래스는 하나 이상의 추상 메서드를 포함하고 있으며
    public void f3(){System.out.println("A.f3()");} //일반 메서드도 가질 수 있다.
    public int getIndex() {
        return index;
    }
    public void setIndex(int index) {
        this.index = index;
    };
}

class BC extends Abstract{
    @Override
    public void f1() {
        System.out.println("f1");
    }
    public void f3(){
        System.out.println("B.f3() and index is " + getIndex());
    }
}

public class AbstractTest {
    public static void main(String[] args) {
        //A a = new A();
        BC b = new BC();
        b.f1(); b.f3();
    }
}
```

- 몸체 없는 메서드를 포함하고 있지 않더라도 abstract 키워드를 포함한 선언 형태
- 추상클래스는 객체를 생성할 수 없다.
- 추상 함수는 자동으로 가상 함수가 된다.
- Overriding은 수정(재구현)의 개념이고 추상클래스는 확장(구현)의 개념

5. 추상클래스 - interface

```
3  interface OpenCloseIf{
4      public void open();
5  }
6  interface PaintIf{
7      public void color();
8  }
9
10 class Door implements OpenCloseIf, PaintIf{
11     @Override
12     public void open() {    System.out.println("open");}
13     @Override
14     public void color() {   System.out.println("color is white");}
15
16 }
17 class Exit implements OpenCloseIf, PaintIf{
18     @Override
19     public void open() {    System.out.println("Don't open");}
20
21     @Override
22     public void color() {   System.out.println("color is red");}
23
24 }
25
26 public class InterfaceTest {
27
28     public static void main(String[] a
29         Door d = new Door();
30         Exit e = new Exit();
31
32         System.out.println("==========
33         System.out.println("Door");
34         d.open(); d.color();
35         System.out.println("==========
36         System.out.println("Exit");
37         e.open(); e.color();
38
```

```
Door
open
color is white
============
Exit
Don't open
color is red
```

```
Public interface Fighter{
    Int ITEM_BOOK = 0;
    Public static int ITEM_NOTE = 1;
        Field영역

    Public void Fighting();
    Public void RunAway();
}
*규칙
1. 생성과 동시에 초기화 필수
2. 기본옵션 public static, 다른 접근제한자 사용X
```

- 계약을 위해서 사용하는 클래스
- 다수의 사람들이 모여서 프로그램 할때 반드시 필요
- 클래스 내의 구현부가 없고 선언부의 집합으로만 이루어진 클래스 (몽땅 추상)
- 접근제한자는 public 디폴트
- 내부에 필드를 가질 수 없다.
- 객체선언은 불가능하나 고정 상수는 가능.
- 멤버에 어떠한 접근자, 한정자 x
- 인터페이스의 목적은 구현이다.

5. 추상클래스 요약

추상클래스 -
클래스로 new 변수(인스턴스화) 생성을 막아야하는 경우가 있는데 그때 사용
추상메서드 -
메서드 구현부를 생략시켜 상속받는 하위 클래스가 강제로 오버라이딩하게 가이드라인을 제공

왜? 위에서 받아쓸 수 있지만 하위에서 기능이 달라지는 경우. 버튼클릭의 동일한 메서드에서 A에서는 열리는 기능 B에서는 소리나오는 경우

인터페이스
자바에서는 다중상속이 허용되지 않는다. 부모가 둘일 수 없다. But, 우리는 두개의 클래스를 하나로 정의해 필요시 상속을 통해 기능을 제공받기를 원합니다.

Q. Interface를 상속 받았을 때 메서드를 강제로 재정의? YES
Q. Abstract에서 메서드를 강제로 재정의? NOPE
Q. Extends는 왜 다중상속 지원이 안된다 해놓고 됨? Interface가 되는거
예를들어, 두개의 interface를 상속관계로 표현하려면 extends를 써야함.
interface는 구현부가 없어서 메서드명이 겹쳐도 문제 될 일이 없다.
그러나 abstract는 내부 구현코드가 다를 수 있어서 문제.

Abstract Class, Class = Class혈통
Interface = Interface혈통 (재정의 필수, 주로 이벤트 핸들러에 사용)
부모 자식간의 혈통이 같으면 extends, 다르면 implements.

6. final 키워드

변수 선언시 그 값을 변화시키지 말아야 할 때 사용된다.

Final class - 상속금지

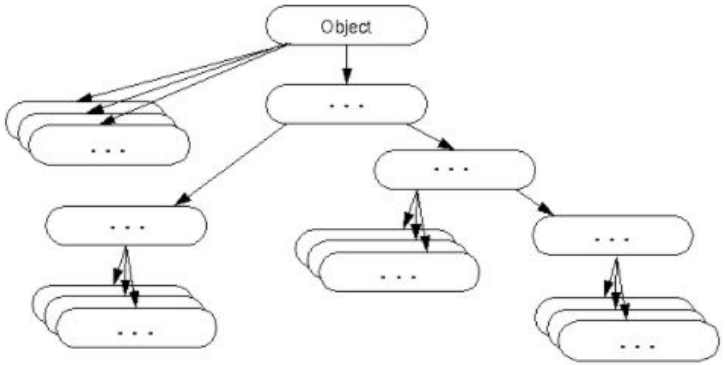
Final Method - 오버라이딩 금지

Final variables - 초기화 금지 (Read-Only)

참고해보면 좋을 문서 :
https://blog.lulab.net/programming-java/java-final-when-should-i-use-it/#fn:4

7. Object 클래스

java.lang.Object 클래스는 자바 API의 모든 클래스와 사용자가 정의한 모든 클래스의 최상위 클래스이다. 즉, 모든 클래스들은 Object 클래스로부터 상속받는다. 다시 말해 Object 클래스의 모든 메서드와 변수는 다른 모든 클래스에서도 사용 가능하다는 말이다.



* Object 클래스의 주요 메소드

메소드	설 명
boolean equals(Object obj)	두 개의 객체가 같은지 비교하여 같으면 true를, 같지 않으면 false를 반환한다.
String toString()	현재 객체의 문자열을 반환한다.
protected Object clone()	객체를 복사한다.
protected void finalize()	가비지 컬렉션 직전에 객체의 리소스를 정리할 때 호출한다.
Class getClass()	객체의 클래스형을 반환한다.
int hashCode()	객체의 코드값을 반환한다.
void notify()	wait된 스레드 실행을 재개할 때 호출한다.
void notifyAll()	wait된 모든 스레드 실행을 재개할 때 호출한다.
void wait()	스레드를 일시적으로 중지할 때 호출한다.
void wait(long timeout)	주어진 시간만큼 스레드를 일시적으로 중지할 때 호출한다.
void wait(long timeout, int nanos)	주어진 시간만큼 스레드를 일시적으로 중지할 때 호출한다.