JAVA Beginning 2

백 성 애

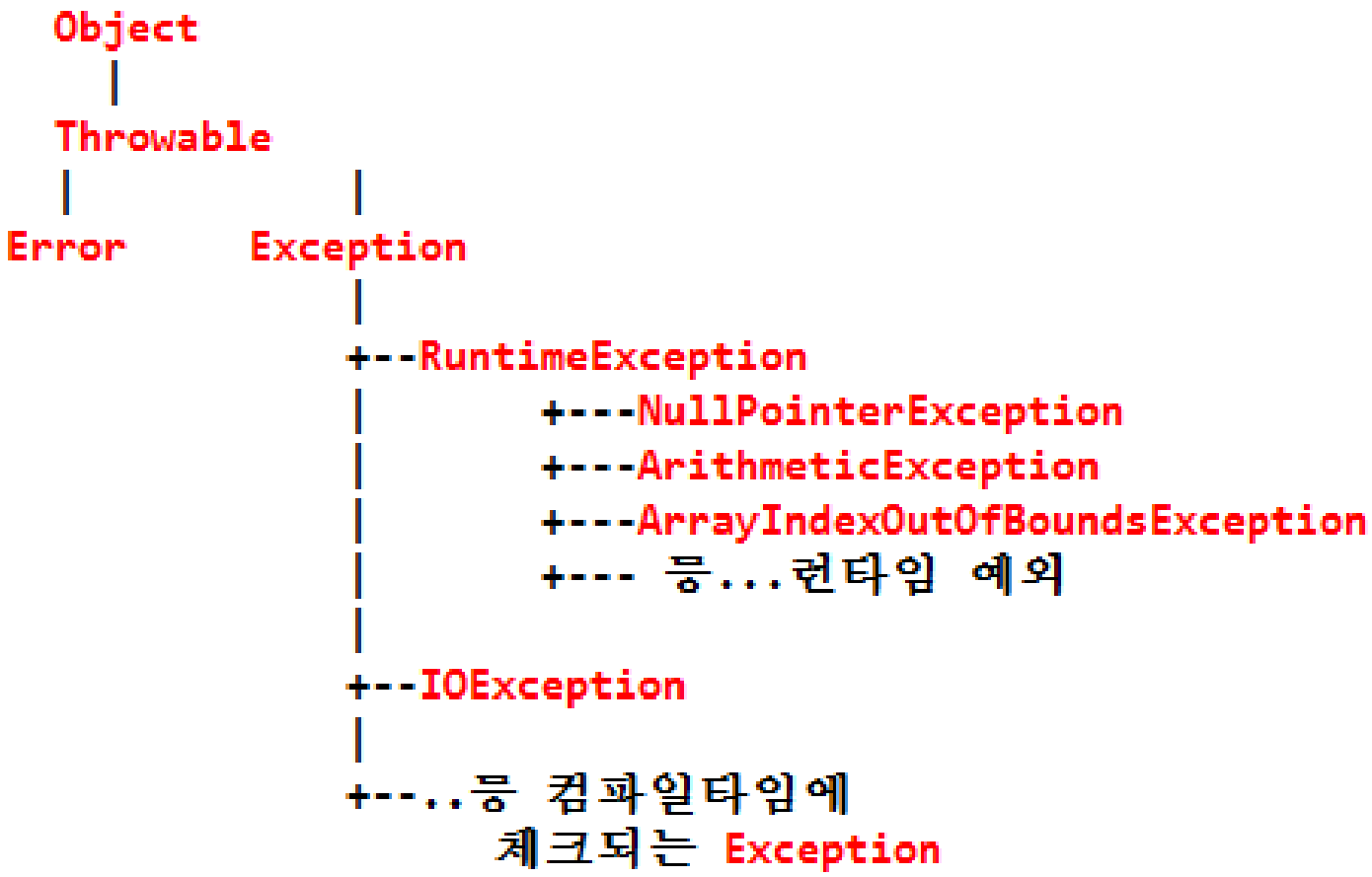
1. Exception

- Exception (예외)

- 1) 예외란?
- …프로그램이 진행하는 과정에서 만나게
- 되는 오류(가벼운 정도의 에러)

1. Exception

- 2) 예외 관련 상속도



3) 예외 처리 목적

...프로그램 진행시 발생할 수 있는 상황들을 미리 정해놓고, 해당하는 예외가 발생했을 경우 적절한 조치를 취해서 프로그램이 정상적으로 작동하도록 하기 위함

4) 예외 처리 방법

1> Handle하는 방법: 직접 처리하는 방법

try~catch 절을 이용

2> Declare하는 방법: 선언하는 방법

throws 절을 이용

1>Handle하는 방법 : try~catch절 이용

-구체적인 예외 처리가 가능하다.

```
try
{
        예외 발생 코드
}
catch (해당예외클래스  e)
{
        예외 처리 코드
}
```

- 2> Declare하는 방법-선언하는 방법
- :throws 절을 이용법
- -예외 처리를 직접 하지 않고 메소드를 호출하는 쪽으로 넘긴다.

ex)

```
public void sub()
throws IOException
{
        int r=System.in.read();
}
```

--->위 처럼 선언하면, sub()를 호출하는 쪽으로 IOException 을 넘겨준다.

5) RuntimeException계열 예외들은******
throws 를 안해줘도 JVM이 알아서
throws 를 해주므로 생략해도 됨.

6) 하나의 메소드에서 두개 이상의 Exception이
발생할 경우
--> catch절을 여러 개 사용할 수 있다.

이 때 주의**********
catch 절로 Exception을 잡을 때는
하위클래스(자식클래스)부터 잡아준다.
****

7) finally 절

...try절이 나와야 쓸 수 있는 구문으로,
try구문이 수행된 후, 또는 catch구문이
수행된 후에 반드시 한번은 수행되는
블럭이다.

[심지어 return 문이 오더라도 finally블럭은
반드시 실행된다.
단 System.exit(0)의 경우
는 실행되지 않고 종료된다.]

예외 관련 문제]
... 다음 결과를 예측해서 출력될 output(결과)를
기재해 보세요.

```
try
{
   1번>...이 부분에서 AException이 발생했다면?

   2번>...이 부분에서 BException이 발생했다면?

   3번>...이 부분에서 CException이 발생했다면?

   4번>...이 부분에서 기타 Exception이 발생했다면?

}catch (AException e){
   System.out.println("AException 발생");
}catch(BException e){
   System.out.println("BException 발생");
   return;
}catch(CException e){
    System.out.println("CException 발생");
    System.exit(1);
}catch(Exception e){
    System.out.println("암튼...예외 발생");
}finally{
    System.out.println("반드시 한번 수행코드");
}
System.out.println("The End~~!");
```

- 8) 사용자 정의 예외 클래스 만들기
- - Exception을 상속받는 클래스를 만든다.
- - 생성자를 구성하고, 생성자 안에서
- super(예외 메시지);를 호출한다.
- -->여기에 들어간 예외 메시지가
- getMessage()를 호출할 때 반환되는
- 메시지가 된다.
- - 옵션: toString()메소드를 오버라이딩해
- 예외 관련 정보를 문자열로 돌려준다.

- ex) class NotKongException extends Exception
- {
- public NotKongException(){
- super("콩씨는 등록할 수 없어요!");
- }
- }//

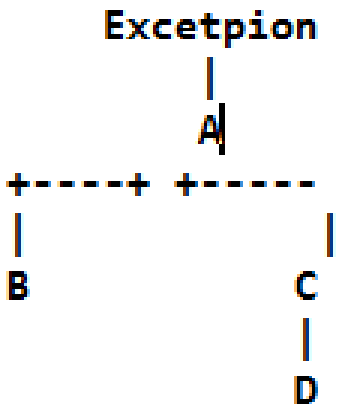
9) 사용자 정의 예외 클래스를 사용할 때는...
위에서 만든 NotKongException을 사용해보자.

```
class Site
{
    public void login(String name)
    throws NotKongException
    {
    if(name.startsWith("콩")){
        throw new NotKongException();
            //throw와 throws가 쌍으로 존재
    }
    }
}//////////////////////////////////
```

사용자 예외 클래스 객체를 생성해서 throw 라는 키워드로 던져주고, 반드시 메소드 헤더부분에 해당 예외를 throws 해줘야 한다.

1. Exception과 Overriding조건

10) Exception과 Overriding 조건




```
//////////////////////////////////////////////////////////////
class Super                          | class Sub extends Super{
{                                                |
                                                 |
    void a() throws C{               |    void a() throws C, D{
    }                                            |    }
}                                                |}
//////////////////////////////////////////////////////////////
- Exception의 경우 부모 클래스의 메소드와
  동일하거나 더 구체적인 Exception을 발생시켜야 한다.
//////////////////////////////////////////////////////////////
```

2. Inner Class
(내부클래스, Nested Class)

```
----------------------------------------
class A
{
	class B
	{//Inner 멤버 클래스: 클래스 안에 선언된 클
	}
	void func(){
		class C//Inner 로컬 클래스
		{
		}//////
	}//func()----------
}//////////////////////
----------------------------------------
```

2. Inner Class

1) 정의
... 클래스 안에 클래스를 구성하는 것
2) 종류
... 클래스가 정의되는 위치에 따른 분류

```
        |
        +--i) Inner Member 클래스
        |          +---a) non-static클래스
        |        +---b) static 클래스
        |
        +--ii)Inner Local 클래스
                 +--a) Named Local Class
                 |     (이름있는 로컬 클래스)
                 +--b) Anonymous Class
                       (이름 없는 클래스,
                          익명클래스,
                        무명안긴 클래스)
```

2. Inner Class

3) Inner 클래스 객체생성 방법 및 사용법
...> Outer.java 예제, Local.java

4) 로컬 이너 클래스는
a) 이름있는 로컬 이너 클래스
b) 이름없는 로컬 이너 클래스
로 구분할 수 있는데, 로컬 클래스는 지역변수와 비슷한 성격을 가지며, 활용범위가 정의되는 메소드 블럭 내부로 제한 된다.

2. Inner Class

로컬 클래스는 외부(Outer)클래스의 멤버변수와, 메소드의 final 로컬변수, 그리고 final 매개변수의 사용이 가능하다.
(로컬 클래스가 메소드의
지역변수 값을 변화시키지 않기 위해서...)

2. Inner Class
-이름있는 로컬 이너 클래스

a)이름있는 로컬 이너 클래스의 경우

- 우선 정의를 한 다음에만 사용할 수 있다.
- 다른 클래스의 상속이 불가능하다.
- 컴파일 하면 OuterClass$숫자+로컬클래스명의 클래스 파일이 생성된다.

  여기서 숫자는 인덱스를 의미하는데, 서로 다른 메소드인 경우 동일 명칭의 클래스가 존재할 수 있기 때문에 중간에 인덱스 역할의 숫자를 붙여 구분되도록 한다.

2. Inner Class
-이름없는 로컬 이너 클래스

b)이름없는 로컬 이너 클래스(Anonymous 클래스)

- 이름을 갖지 않는 이너 클래스
- 한번만 객체 생성이 가능하다.
- 객체를 생성하는 문장 뒤에 클래스의 블럭을 덧붙이는 방법으로 구성
- new 키워드 뒤 생성자의 명칭이 기존 클래스 명일 경우에는 Anonymous클래스가 자동적으로 클래스의 자식 클래스가 되며, 인터페이스일 경우에는 이 인터페이스를 상속하는 클래스로서, 부모 클래스가 Object이 된다.

3. java.util.Vector 클래스

- 객체를 저장할 수 있는 클래스
  - 배열과 기능면에서 유사

  int a[]=new int[3];

  Student st[]=new Student[3];

  cf> 배열->기본자료형, 참조형 저장 가능
    - 같은 타입의 데이터만 저장
      - 고정 크기 : 일단 생성된 후에는 배열 크기를 변경하기 불가능

        int a[]=new int[3];

반면 Vector 는

ex) Vector v=new Vector(3, 2);

//3-> 용량, 2->증가치

3. java.util.Vector 클래스

- Object 유형의 Collection 으로서,
  어떤 유형의 객체도 저장할 수 있다.
  [단, 기본자료형은 저장 불가.]
  - 벡터는 데이터가 가득 차면 자동으로
    저장 영역을 확대한다.
  - java.util.List 인터페이스를 구현하는 클래스로
    순서대로 정렬되고, 중복도 허용하는 특징
    을 갖는다.

3. java.util.Vector 클래스

1) Vector 클래스 생성자

1> Vector()//10개 요소를 저장할 공간

2> Vector(int initialCapacity)//초기치 용량을 매개변수로 정할 수 있다.

3> Vector(int initialCapacity, int incre)

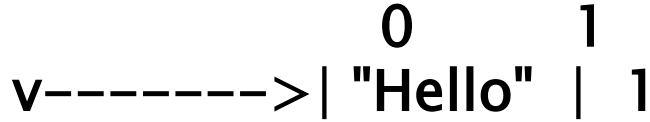
: 초기 용량과 증가치를 인자로 넣어줌

2) Vector에 요소 저장

ex) Vector v=new Vector(5);

v.add("Hello");

v.add(new Integer(10));



1> add(Object obj)

2> addElement(Object obj)

3. java.util.Vector 클래스

3) Vector에서 꺼내오기

ex) Object obj= v.get(0);
String str=(String)obj;

Integer i=(Integer)v.get(1);

1> Object get(int index)
2> Object elementAt(int index)
3> Enumeration elements();

3. java.util.Vector 클래스

4) elements()/iterator()로 벡터 요소 꺼내오기.

- Enumeraiont과 Iterator인터페이스는 객체들을 집합체로 관리하는데, 이들 인터페이스에는 각각의 객체들을 한순간에 하나씩 처리할 수 있는 메소드를 제공한다.

-Vector의 메소드 : Enumeration elements()

```
/**********************************************
Enumeration인터페이스에 있는 주요 메소드
  1>boolean hasMoreElements()
  2> E nextElement()
**********************************************/
```

3. java.util.Vector 클래스

-Vector의 메소드 : Iterator iterator()

```
/**********************************************
Iterator 인터페이스의 주요 메소드
  1>  boolean hasNext()
  2> E next()
  3> void remove()
**********************************************/
```

**java.util.객체 저장 클래스

1) Collection 계열

Collection

| |

Set List

|

SortedSet

1>특징

- 객체 입력 순서를 기억
- 중복을 허용

2> 하위의 주요 클래스

- ArrayList, Vector

2) Map 계열

Map

|

SortedMap

1> 특징

-Object유형(객체형)의 key와 value가 매핑(mapping)되어 저장

- key값은 중복되어선 안된다. [value는 중복 허용]

2> 하위의 주요 클래스

-Hashtable, Properties

4. AWT 이벤트 처리

1) 이벤트(Event)란?

...프로그램과 관련있는 사건

ex)- 사용자가 버튼을 누르는 사건

- 마우스를 움직이는 사건

- 텍스트필드에서 입력후 엔터를 치는 사건...등...

4. AWT 이벤트 처리

2) 이벤트 처리를 위한 기본 개념

i) 이벤트소스(Event Source) :
ii) 이벤트(Event)
iii)이벤트 처리자(Event Handler)

**위에서 프로그래머가 신경 쓸 부분은
Event Source 와 Event Handler 이다.
Event 는 JVM이 알아서 객체를 생성해
발생시킨다.

4. AWT 이벤트 처리

i) Event Source

– 이벤트를 발생시키는 근원지

..컴포넌트나 컨테이너가 이벤트 소스가 됨 ex) Button, TextField, Panel 등...

ii) Event

– 이벤트 소스에 따라 발생되는 Event가 달라진다.

즉,

Button 일 경우 ActionEvent가 발생되며

Checkbox일 경우는 ItemEvent가 발생된다.

..각 이벤트 객체는 JVM이 발생시킨다..

4. AWT 이벤트 처리

iii) Event Handler

- Event를 처리해주는 클래스를 의미한다.
- 프로그래머가 직접 구현해주어야 하는 클래스이다.
- 이벤트 핸들러가 되기 위해서는 XXXListener란 인터페이스를 상속받아야 한다.

ex)-Button 일 경우 --> ActionEvent 발생
---> ActionListener 를 상속받는 클래스를 구성해야 함.

-Checkbox --> ItemEvent 발생
----> ItemListener 를 상속받는 클래스 구성

4. AWT 이벤트 처리

- 각각의 리스너를 상속받은 클래스에서는
  의무적으로 추상메소드를 재정의해야 한다.
  이 메소드가 바로 이벤트를 처리해주는
  메소드가 된다.
  ex) ActionListener -> actionPerformed()
  ItemListener -> itemStateChanged()

****

이벤트 소스가 결정되고
이벤트 핸들러를 구성했다면
이벤트 소스와 핸들러를 연결해줘야 한다.
-->addXXXListener()메소드를 이용
ex)bt.addActionListener(핸들러객체)

****

4. AWT 이벤트 처리

3) Event 처리 절차

1> import java.awt.event.*;
2> XXXListener 인터페이스를 상속
3> 2번이 가지고 있는 추상 메소드 재정의
4> 이벤트소스와 핸들러를 연결해줌
...addXXXListener()메소드로.
5> 3번에서 재정의한 메소드에 구체적인
이벤트 처리 코드 구현

4. AWT 이벤트 처리

4) Event 핸들러를 구성하는 방법

1> 이벤트소스(컴포넌트)를 가지는 자기 클래스가 핸들러가 되는 방법
2> 이벤트 핸들러를 내부클래스(Inner class)로 별도로 구성하는 방법
3> 이벤트 핸들러를 외부클래스로 구성하는 방법

5) 이벤트 관련 Interface와 Adapter클래스

이벤트	인터페이스	추상메소드	어댑터 클래스
ActionEvent	ActionListener	actionPerformed()	X
ItemEvent	ItemListener	itemStateChanged()	X
Adjustment Event	AdjustmentListener	adjustmentValue Changed()	X
WindowEvent	WindowListener	windowClosing() windowClosed() widnowOpened() windowIconified() windowDeiconified() windowActivated() windowDeactivated()	WindowAdapter
MouseEvent	MouseListener	mouseEntered() mouseExited() mouseClicked() mousePressed() mouseReleased()	MouseAdapter
	MouseMotionListener	mouseDragged() mouseMoved()	MouseMotionAdapter

5. paint()/repaint()/update() 메소드 관계

1) paint()메소드 : 프로그래머가 직접 호출
할 수 없다.
JVM이 호출해주는 메소드
프로그래머는 오버라이딩만
할 뿐...

그러나 paint()작업을 다시 해야 할 때가 있다.
그럴 때는 repaint()메소드를 호출하자.

5. paint()/repaint()/update() 메소드 관계

2) repaint()메소드: 프로그래머가 호출할 수 있는 메소드.
repaint()를 호출하면,JVM이 자동으로 paint() 메소드를 호출해준다.

3) update()메소드 :
프로그래머가 repaint()를 호출하면 JVM은 paint()를 호출하기에 앞서, update()메소드를 호출한다.

update()메소드에서 하는 일은
i) 배경색으로 모두 지워준 뒤
ii) paint()를 호출하는 일을 한다.

5. paint()/repaint()/update() 메소드 관계

즉..repaint()를 호출하면

repaint()-->update(Graphics g)
+--->지우기-->paint(g)
개발자--->| JVM--->

java.lang.Thread 클래스

1)Process 란?

...컴퓨터 내에 실행 중인 프로그램을 의미
프로세스는 각각 독립된 주소공간을 가지고
실행되며, 서로 독립적으로 실행되도록
스케줄링 할 수 있다.

2) Thread 란?

...하나의 프로세스 안에서 실행되는 명령 흐름
(프로세스 안의 작은 프로그램)
스레드는 자신의 주소공간을 갖지 않고,
실행될 때 그 프로세스의 메모리와 자원을
사용한다.

java.lang.Thread 클래스

- Multi Tasking : 동시에 word, exel, jvm이
  작업하는 것처럼 보이는 것.

- Multi Processing: 아예 동시에 작업 실행하는 것.
  일반적인 개인용 컴퓨터에서는
  cpu가 하나만 들어있지만,
  중대형 컴퓨터에서는 여러 개의
  cpu를 사용하는 경우가 많다.
  여러 개의 cpu에서 동시에
  여러 개의 프로세스가 수행되
  는 것을 멀티 프로세싱이라 함

java.lang.Thread 클래스

- Multi Threading : 프로세스 안에서 여러 개의 스레드가 동시 작업하는 것.
즉 한 프로그램 내에서 두 가지 이상의 일을 수행하는 것으로서,

ex) 워드를 치면, 자동교정,
자동 저장, 타이핑 등
작업이 동시에 이뤄지는데
이는 멀티 스레딩의 예.

java.lang.Thread 클래스

3) 스레드를 작성하는 방법

1> java.lang.Thread 클래스를 상속받아 구현
2> java.lang.Runnable 인터페이스를 상속받아 구현

1>Thread를 상속받는 경우
....run()메소드 오버라이딩.

```
class SnailThread extends Thread
{
        public void run(){
                //스레드가 할 일을 구성

        }
}//////////////////////////////////
```

java.lang.Thread 클래스

**SnailThread를 사용하는 응용프로그램에서는 아래와 같이 사용한다.

```
SnailThread tr=new SnailThread();
  tr.start();

//run()을 호출하는 것이 아니라,
//start()를 호출함으로써 Thread가 실행된다.
```

java.lang.Thread 클래스

2> Runnable을 상속받는 경우

...역시 run()메소드를 오버라이딩

```
class Snail implments Runnable
{
    public void run(){
        // 스레드가 할 일을 구성
    }
}////////////////////////////
```

**응용 프로그램에서는 아래와 같이 사용

```
Snail r=new Snail();
Thread tr=new Thread(r);
tr.start();
```

java.lang.Thread 클래스

- 스레드를 상속받는 경우:
- 클래스와 무관하게 독립된 작업을 하거나, 여러 개의 스레드가 필요한 경우에 적합하다.
- Runnable을 상속받는 경우:
- 클래스 특성을 그대로 유지 하면서 스레드를 하나씩 생성할 수 있다는 장점이 있다.

java.lang.Thread 클래스

4) Thread의 주요 메소드

1> start() : start했다고 해서 바로 run하는 것이 아님
단지 run을 준비하는 상태로 들어감
2> activeCount()
3> currentThread()
4> setName()
5> getName()
6> sleep() : 자기보다 낮은 순위의 스레드에게 실행기회를 준다.

java.lang.Thread 클래스

7> yield() : 자기와 우선순위가 같거나 높은 스레드에게 실행기회를 준다.

8> join() : 칠판 설명을 참조..

9> interrupt(): sleep()및wait()메소드가 수행중일 때, 이 메소드를 호출하면 interruptedException이 발생된다.

10> wait() : 동기화 블럭

11> notify() : 동기화 블럭

12> notifyAll() : 동기화 블럭

13> stop(): 스레드 강제종료-Depricated됐으므로 사용하지 말자.

java.lang.Thread 클래스

14> setPriority()
15> getPriority()
16> isAlive() : alive는 스레드가 run상태에 있다는 것을 의미하는 것이 아님. 단지 스레드가 start되었고, stop되거나 run off되지 않았다는 것을 의미.

5) Thread 의 주요 상수
1) Thread.MAX_PRIORITY : 10
2) Thread.MIN_PRIORITY : 1
3) Thread.NORM_PRIORITY: 5

java.lang.Thread 클래스

6). 주의 사항

- wait()/notify()/ notifyAll() 메소드는 synchronized 2블럭이나 코드에서 수행되어야 한다.

synchronized(동기화)란?*

...코드의 한 부분이 다른 스레드와 동시에 수행 될 수 없게 하는 것.

synchronized 란 키워드는 modifier의 일종으로 동기화 블럭을 만들때 사용한다.

java.lang.Thread 클래스

7). 스레드를 종료시키려면?

1> stop() 호출: 강제종료 -> 사망
1)의 방법은 문제점이 있으므로 지양
2> interrupt()메소드를 사용해서 처리하는 방법
3> 상수를 이용하는 방법
--> 스레드를 이용한 애플릿 예제 참조

java.lang.Thread 클래스

8)스레드 동기화(synchronized);

...어느 주어진 시간에는 하나의 스레드만이 동일한 자료를 참조할 수 있도록 제한하는 방법

-synchronized 는 메소드에 사용이 가능한 modifier 로서, 여러 스레드에 의해 특정 객체의 메소드들이 동시 호출되는 것에 대해 잠금(lock)을 설정하여, 거부하도록 하는 기능을 지원한다.

- 즉 코드의 한부분, 또는 동기화된 메소드가 다른 스레드와 동시에 수행될 수 없게 하는 것

java.lang.Thread 클래스

9) synchronized 구현 방법

1> synchronized 블럭을 이용

```
ex) public void push(char c){
                    ////////
              synchronized(this){
                    동기화 블럭

              }////////////////////
}//----------------------
```

java.lang.Thread 클래스

2> 메소드 앞에 synchronized 지정자를 붙임

ex)

```
synchronized public void push(char c){
        동기화된 메소드
  }//--------------
```

...2>의 방법을 이용할 경우, 메소드 전체가 동기화 블럭이 되어, 필요 이상으로 엄격하게 lock flag를 유지하고 있는 결과를 초래.

java.lang.Thread 클래스

10) wait()/ notify() /notifyAll() 메소드
...동기화 블럭에서 수행되는 메소드
-> 그렇지 않으면 IllegalMonitorStateException 발생
- wait() 메소드 사용시 InterruptedException 이 발생할 수 있으므로, 예외 처리를 해줘야 함.

1> wait() 가 호출되면...
- 수행 권한을 포기한다.
- 모니터 락(lock)을 반납한다.
- 대기상태(Waiting Pool)로 들어간다.

java.lang.Thread 클래스

2> notify()가 호출되면...

- Waiting Pool에 들어가 있는 스레드 하나를 Runnable 한 상태로 전환시킨다.
- notify()는 특정 스레드를 깨워주는 것이 아니라, wating pool에 있는 아무 스레드나 깨울 수 있음
- 반면 notifyAll() 메소드는 waiting pool에 있는 모든 스레드를 깨워준다.

java.lang.Thread 클래스

11) DeadLock 이란?

... 두 스레드간의 상호 의존에 기인하는 버그
다중 스레드가 다중 자원에 접근하기 위해 경쟁하는 프로그램에서 DeadLock이 발생할 수 있다.

ex) 칠판 그림 참조...
설계 단계에서부터 DeadLock을 방지하는 설계를 해야 함.