

---

# Java Advanced

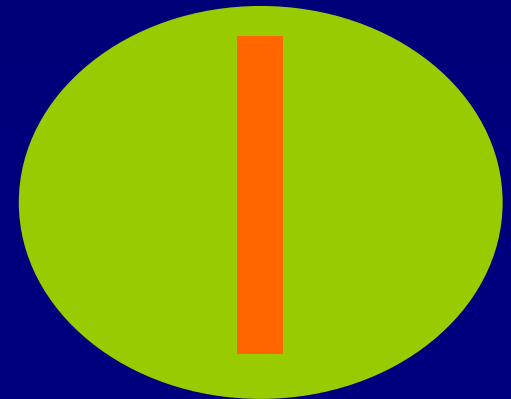
---

# Thread

# Thread

---

- 프로세스 내의 개별적인 실행 흐름
- 프로세스는 프로그램 실행에 필요한 자원과 thread로 구성
- 모든 프로세스는 최소 하나 이상의 thread가 존재해야 실행가능



# MultiThread Program

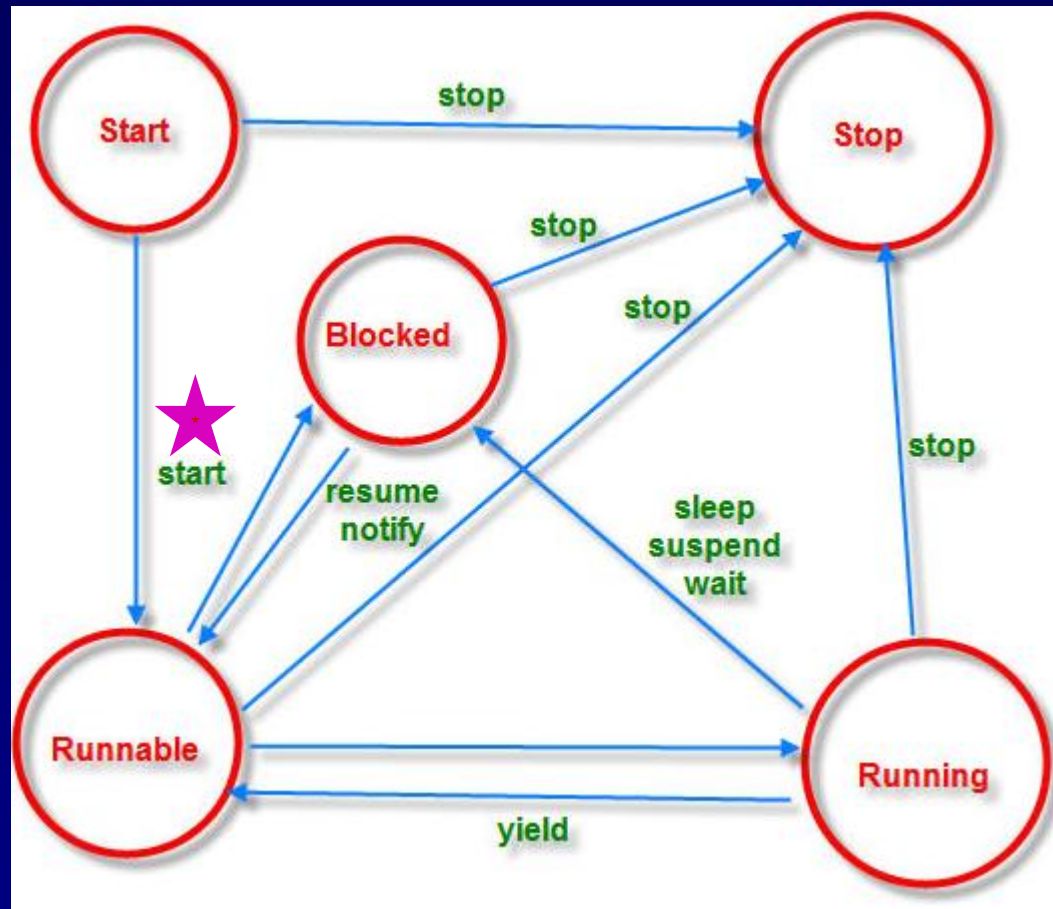
---

- 둘 이상의 thread를 이용한 프로그램
- CPU, 자원 이용률을 높임
- 사용자 응답성이 향상됨



# Thread life cycle

---



# Thread 생성방식

---

1. Thread class 상속
2. Runnable interface 구현

→ 두 방법 모두 run() 구현해야 함  
; thread에 시킬 작업 내용 기술하는 곳

# Thread 생성방식

---

- Thread class 상속

```
class Test extends Thread{}
```

```
Test t = new Test();
```

```
t.start();
```

# Thread 생성방식

---

Thread

Lion

- Runnable interface 구현

Runnable

```
class Test2 implements Runnable{
```

```
Test2 t2 = new Test2();
```

```
Thread thread = new Thread(t2);
```

```
thread.start();
```



# Thread Synchronization

---


- 다수의 thread가 공유 데이터에 접근해서 작업할 경우 thread간의 간섭 현상이 발생
- 이를 방지하기 위해 thread간에 동기화(synchronization)를 시켜 줌  
→ 일종의 lock

# Thread Synchronization

---

```
int count = 0;  
Thread a, b, c;  
public void run(){  
    add();  
}  
public void add(){  
    count++; // problem!  
    System.out.println(count);  
}
```

public synchronized void add()

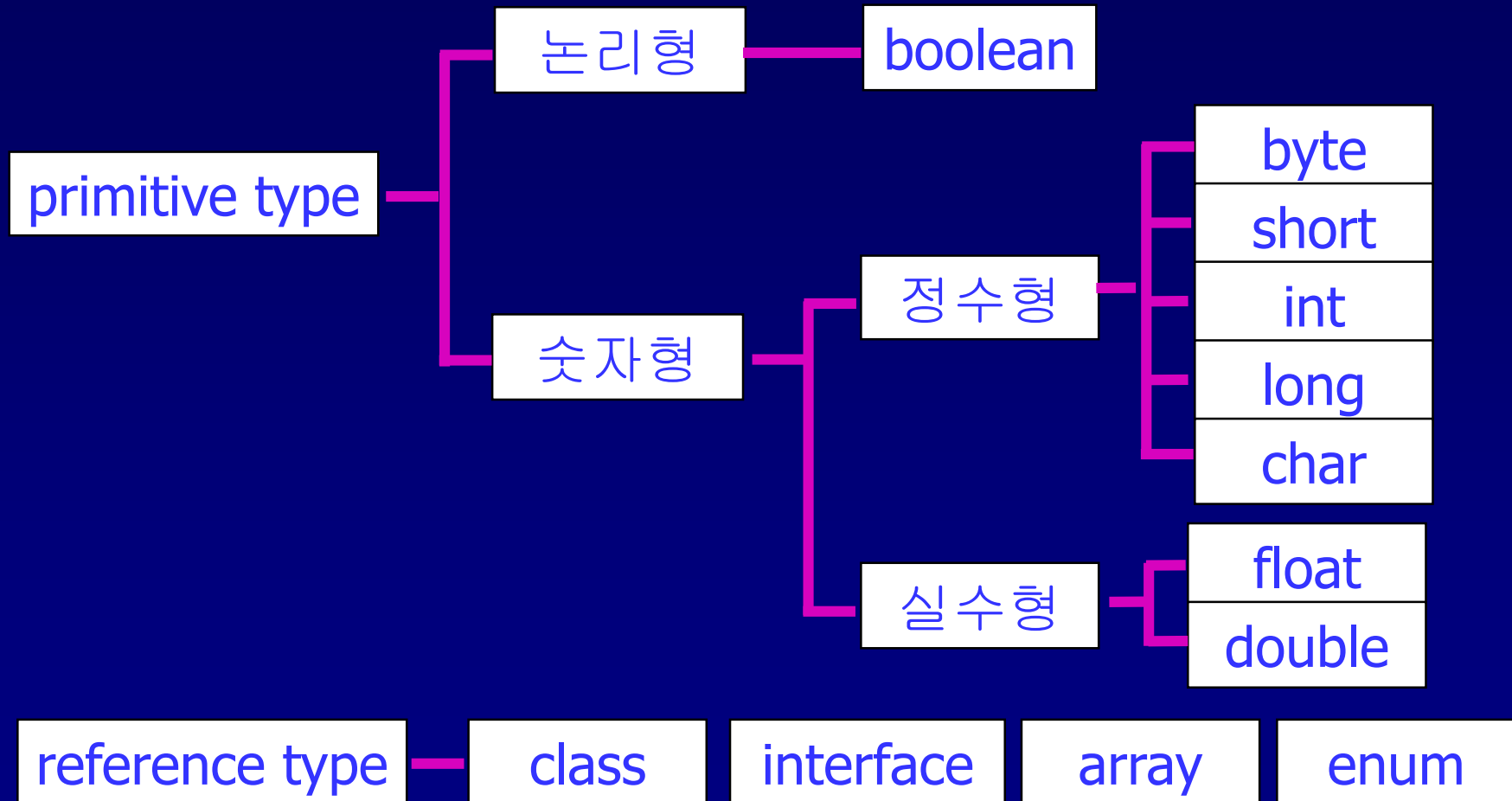


---

# Wrapper Class

# Java Data Types

---



# Wrapper class

---

- 기본형의 데이터를 참조형으로 변환시켜 주는 클래스
- 각각의 기본형 타입마다 해당하는 wrapper class가 존재

# Wrapper class

---

기본형	Wrapper class
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character

---

# Java Collection API

# JCF(Java Collection Framework)

---

- 여러 개의 데이터를 저장하고 관리할 수 있는 자료구조
- 데이터 저장 / 관리 방식에 따라 List/Set/Map구조로 나뉨



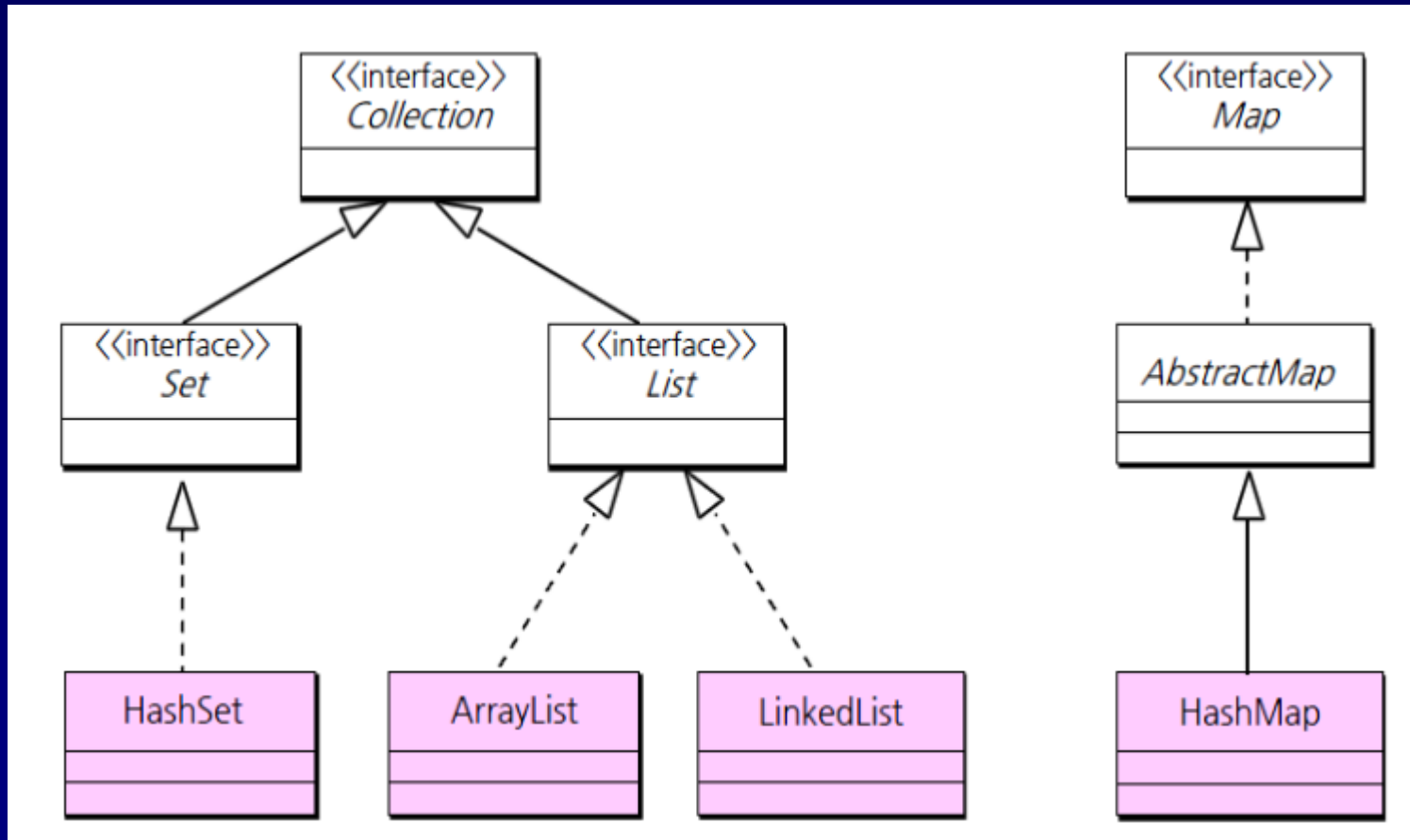
# Collection 관련 클래스

---

List	Set	Map
순서 O 중복 O	순서 X 중복 X	(Key, value)
Vector ArrayList	HashSet	HashMap HashTable

Vector     ArrayList  
HashTable     HashMap     .

# Collection 관련 클래스 상속도



extends (     )  
가     .

---

# Generic Programming

# Generic Programming

---

- 하나의 값이 여러 개의 서로 다른 데이터 타입을 가질 수 있도록 해주는 기술
- 데이터가 특정한 형식에 고정되지 않음
- 재사용성을 고려한 프로그래밍 방식

# Generic Code

---

```
public class Box<T> { //T: type parameter
    private T data;

    public void set(T data) {
        this.data = data;
    }

    public T get() {
        return data;
    }
}
```

# Generic Code

---

Runtime시에 데이터 타입 결정

- `Box<String> b = new Box<String>();`
- `Box<Integer> i = new Box<Integer>();`
- `Box<String> b2 = new Box<>();`

# Multiple type parameter

---

```
interface Pair<K, V> {  
    public K getKey();  
    public V getValue();  
}
```

# Multiple type parameter

---

```
public class OrderedPair<K, V> implements
Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }
}
```



# Type parameter의 제한

---

```
public class C1<T extends Number> {  
    ...  
}
```

```
public class C2<T extends Person &  
Comparable> { ... }
```

---

# **GUI (Graphic User Interface)**

# GUI

---

## AWT

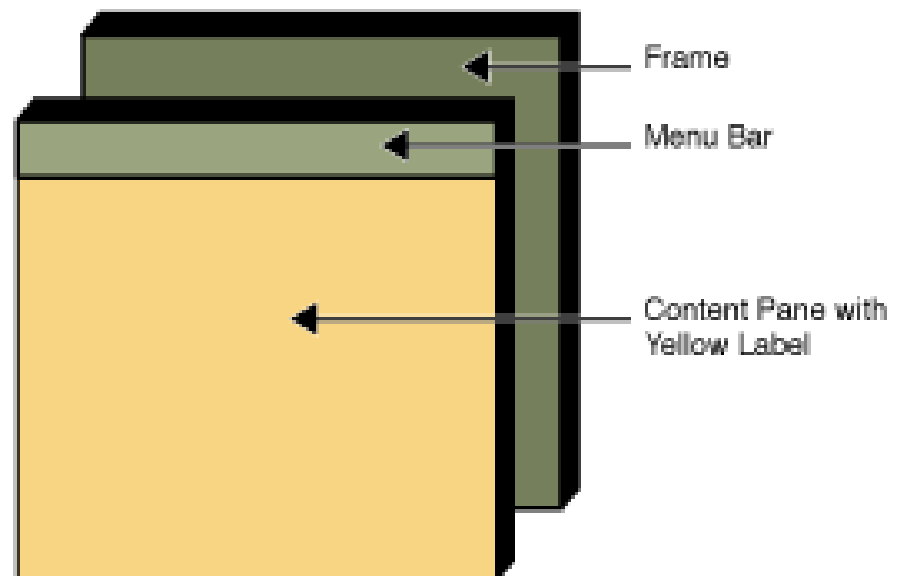
- JDK 초기 UI API
- 컴포넌트가 C와 같은 NATIVE CODE를 포함하고 있음

## SWING

- 다양한 종류의 컴포넌트
- 컴포넌트가 100% JAVA로 작성되어 짐
- 어떤 실행 환경에서든 동일한 UI

# JFrame의 구성

---



# Layout Managers

---

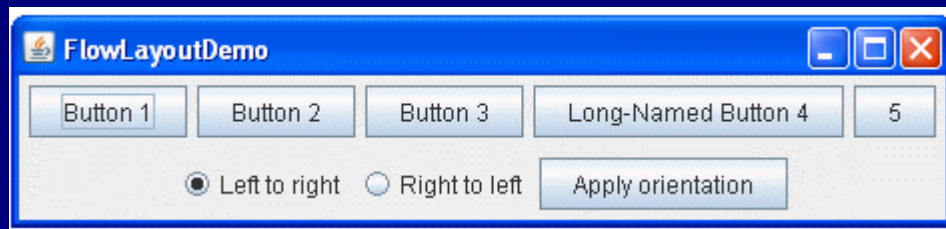
- FlowLayout
- BorderLayout
- GridLayout
- GridBagLayout
- CardLayout
- BoxLayout

more...

# FlowLayout

---

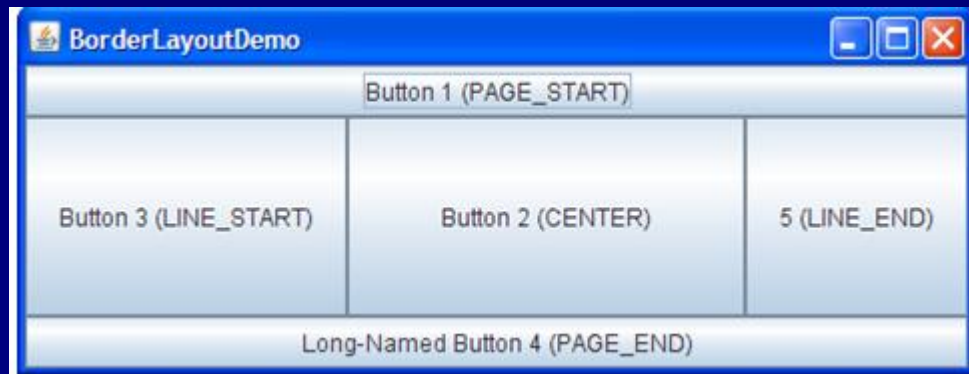
- 화면의 상단에서부터 중앙 정렬되어 오른쪽 방향으로 배치됨
- 화면 크기가 변하면 배치가 달라짐



# BorderLayout

---

- 화면이 동,서,남,북,중앙으로 나누어져 있음
- 화면 크기가 변해도 배치가 달라지지 않음



# GridLayout

---

- 화면이 행렬 방식의 격자로 나누어져 있음
- 화면 크기가 변해도 배치가 달라지지 않음





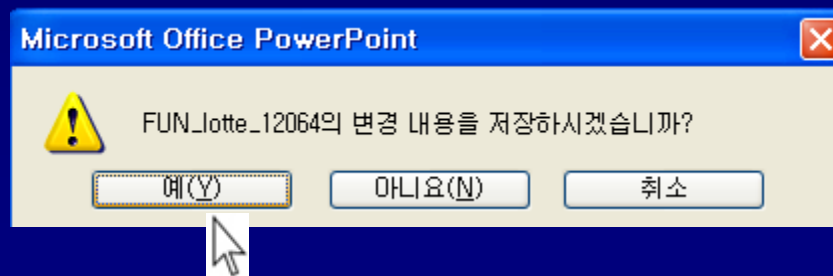
---

# Event Handling

# Event Handling

---

- 화면을 구성하고 있는 컴포넌트들에 대해 발생하는 사건(이벤트)에 대한 처리작업



# Event Handling

---

- Event Handler
  - Event 감시/처리자
  - Interface 타입으로 존재 → 추상메소드
  - Handler 이름 → ~Listener
  - java.awt.event / javax.swing.event

# Event Handling

---

Component	Event	Listener
JButton, JTextField	ActionEvent	ActionListener
JList	ListSelectionEvent	ListSelectionListener
Window	WindowEvent	WindowListener

# Event Handling 처리순서

---

1. 이벤트 처리 클래스 정의  
→ Listener(감시자) implements 하기
2. Event발생 컴포넌트에 Listener 등록
3. Listener의 추상 메소드 구현  
→ Event 발생시 원하는 처리작업 기술