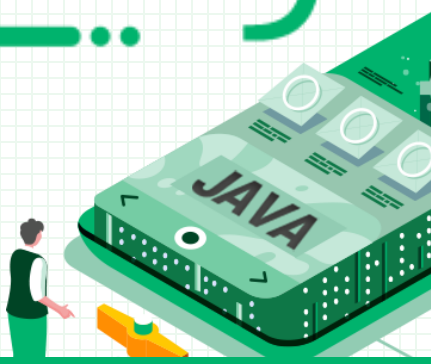
안드로이드 프로그래밍을 위한 자바기초



⑩ 확장 문법 이해하기

한국기술교육대학교
온라인평생교육원

학습목표

- 제네릭을 이용하여 프로그래밍을 할 수 있다.
- 확장 for문 및 열거형 클래스를 이해하고 프로그래밍에 활용할 수 있다.
- 멀티 스레드를 이해하고 프로그래밍에 활용할 수 있다.

학습내용

- 제네릭 이해하기
- 확장 for문 및 열거형 이해하기
- 멀티 스레드 이해하기

제네릭 이해하기

제네릭의 개요

1) 제네릭(generic)이란?

- 클래스나 메소드에서 처리할 자료형을 미리 지정하지 않음
- 클래스의 객체를 생성할 때 자료형을 지정함
- 주로 컬렉션 클래스와 인터페이스 및 반복자 클래스에 사용됨

2) 제네릭 사용의 장점

(1) 하나의 클래스를 자료형 지정을 통하여 해당 자료형 전용 클래스로 사용
(2) 객체 형 변환을 하지 않아도 되어 소스가 간결해짐

3) 제네릭 정의 방법

(1) 꺽쇠표(<, >) 기호를 사용하여 영문 대문자 하나의 글자로 지정

✓ 예 : <T> <E> 등

(2) T와 E등을 자료형 인자라고 하고, 다른 문자를 임의로 사용하여도 됨

제네릭의 활용

1) 제네릭 사용하기

ArrayList클래스의API문서

```
public class ArrayList<E> {
  boolean add(E e)
  Iterator<E> iterator()
}
```

```
public class ArrayList<String> {
  boolean add(String e)
  Iterator<String> iterator()
}
```

```
ArrayList<String> al = new ArrayList<String>();
al.add( new String("AAA") );
Iterator<String> it = al.iterator();
```

제네릭 이해하기

2) 제네릭 클래스 정의하기

```
public class Member<T> {
  T name;
  public Member() {
  }
  void add(T name) {
    this.name = name;
  }
  T get() {
    return name;
  }
}
```

String 전용 클래스

```
Member<String> m = new Member<String>();
m.add( new String("AAA") );
String result = m.get();
```

Integer 전용 클래스

```
Member<Integer> m = new Member<Integer>();
m.add( 200 );
Integer result = m.get();
```

3) 제네릭 제한

제네릭 지정	설명
< ? extends T >	T와 T의 하위 클래스들 지정 가능
< ? super T >	T와 T의 상위 클래스들 지정 가능
< ? extends Object > 또는 < ? >	모든 클래스 지정 가능
< T extends Robot >	Robot 클래스의 하위 클래스만 자료형 인자 T로 지정 가능

실습

제네릭 실습

실행 화면

```
[제네릭 : String 전용 ArrayList]
AAA
BBB
[제네릭 : Integer 전용 ArrayList]
100
200
[제네릭 클래스 정의]
Member<String> : CCC
Member<Integer> : 500
[제네릭 제한 <T extends Toy>]
[Box<Toy>]
Hello! Toy : Hello! Robot : Hello! Car :
[Box<Robot>] …
```

- 소스 파일명 : [ArrayBasic.java]
- 자세한 내용은 실습 영상을 확인해보세요.

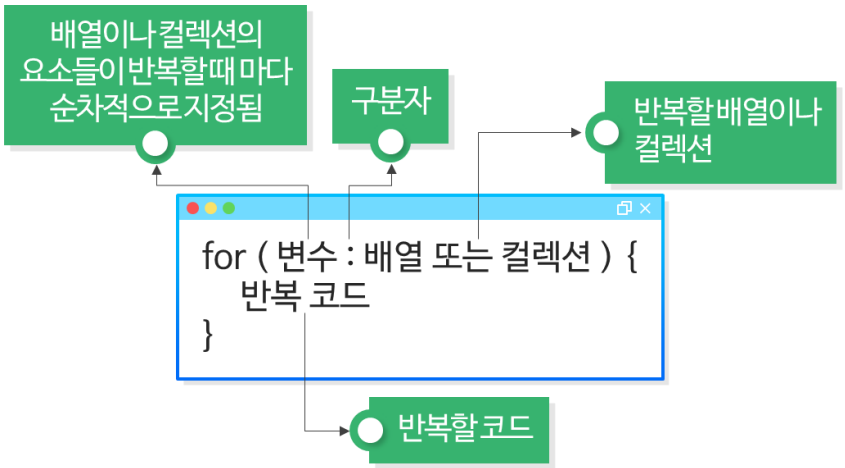
확장 for문 및 열거형 이해하기

확장 for문 이해하기

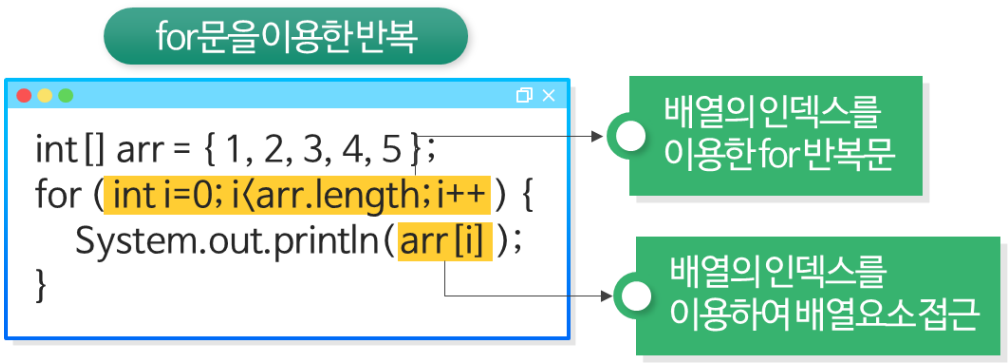
1) 확장 for문이란?

- 배열 또는 컬렉션을 위한 for문
- 배열의 인덱스나 컬렉션의 반복자 없이도 반복 가능

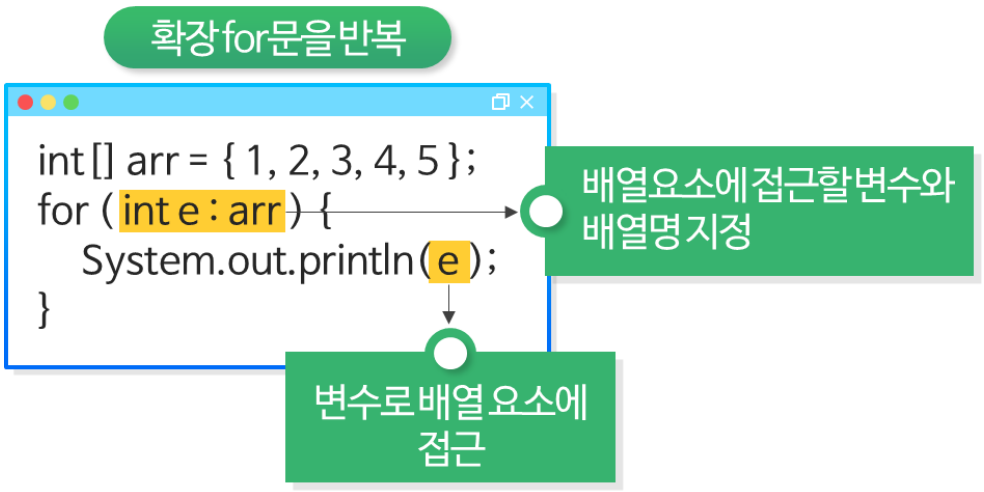
2) 구조



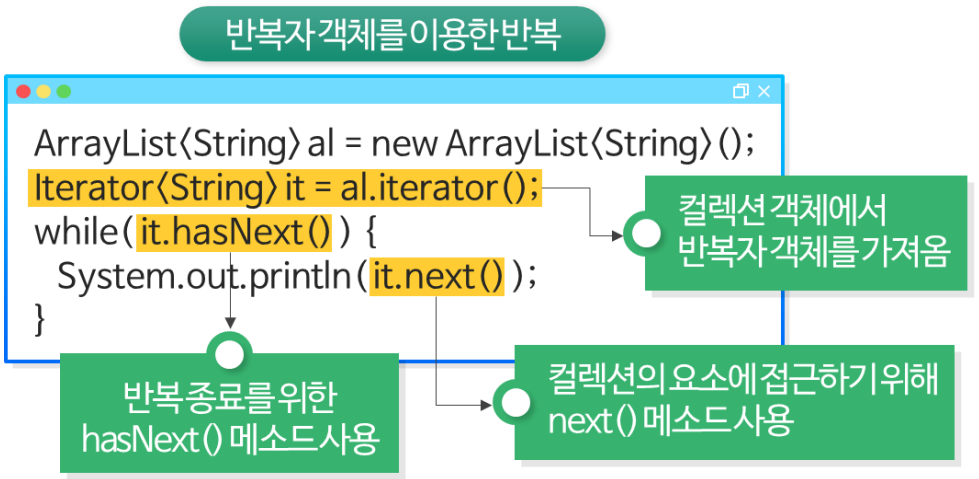
3) 배열에 확장 for문 사용하기

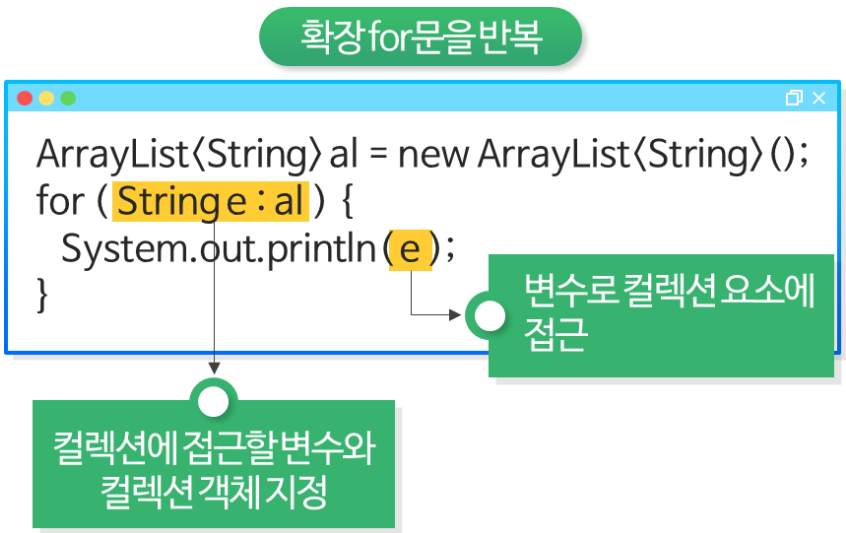


확장 for문 및 열거형 이해하기



4) 컬렉션에 확장 for문 사용하기





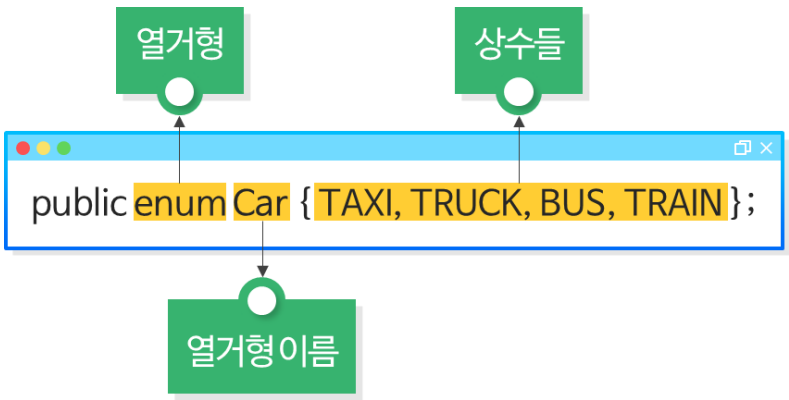
확장 for문 및 열거형 이해하기

열거형 이해하기

1) 열거형(enumeration type)이란?

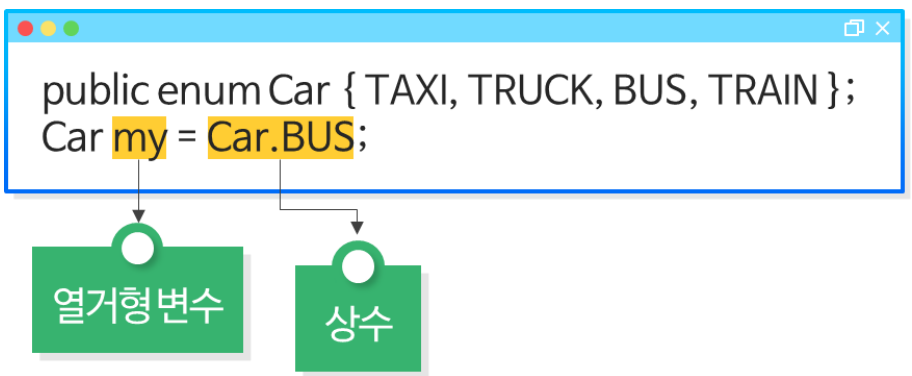
- 상수들을 묶어서 정의한 자료형
- 열거형 변수는 정의된 상수들만 지정할 수 있음

2) 열거형 정의



3) 열거형 사용

(1) 클래스가 아니므로 객체 생성을 할 수 없음
(2) 열거형 변수를 선언하여 사용



실습

확장 for문 및 열거형 실습

실행 화면

```
[배열 : for문]
10
20
30
40
50
[배열 : 확장 for문]
10
20
30
40
50
[컬렉션 : while문] …
```

- 소스 파일명 : [ExtendFor.java]
- 자세한 내용은 실습 영상을 확인해보세요.

멀티 스레드 이해하기

멀티 스레드(multi-thread)의 개요

1) 프로세스(process)란?

- 실행중인 프로그램
- 하나의 프로세스는 한 개 이상의 스레드로 동작됨

2) 스레드(thread)란?

- 실제로 작업을 수행하는 단위

(1) 싱글 스레드 : 프로세스가 하나의 스레드로 동작
(2) 멀티 스레드 : 프로세스가 동시에 두개 이상의 스레드로 동작

3) 멀티 스레드의 예

상대방의 채팅내역이
화면에 표시되면서 동시에
내가 입력한 채팅내역을 전송

사용자의 동작도 수행하면서,
상대방의 동작도 동시에 수행

멀티 스레드 이해하기

멀티 스레드 구현 방법

1) 멀티 스레드 클래스 구현

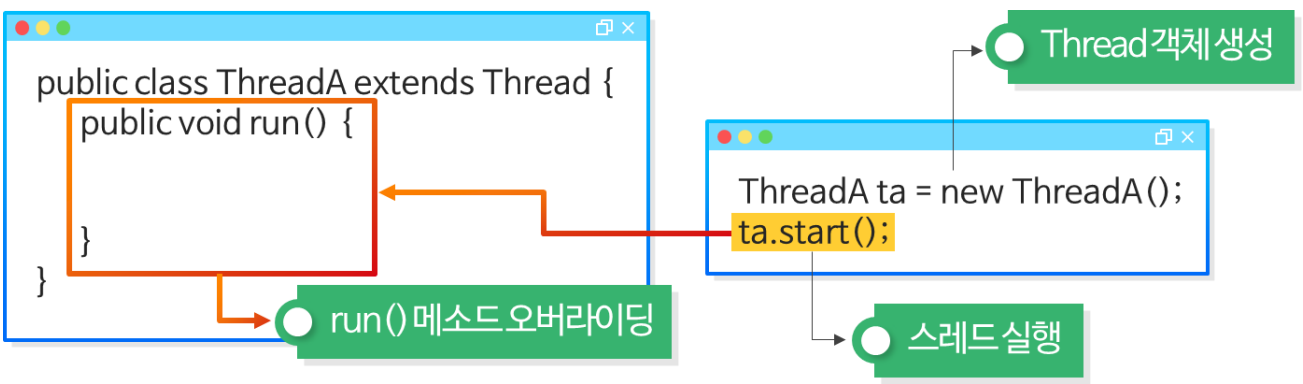
(1) Thread 클래스를 상속
(2) Runnable 인터페이스를 구현

2) run() 메소드 오버라이딩

(1) 무한 반복문을 사용하여 프로그램 작성
(2) start() 메소드를 호출하면 run() 메소드가 실행됨

3) Thread 클래스 상속 방법

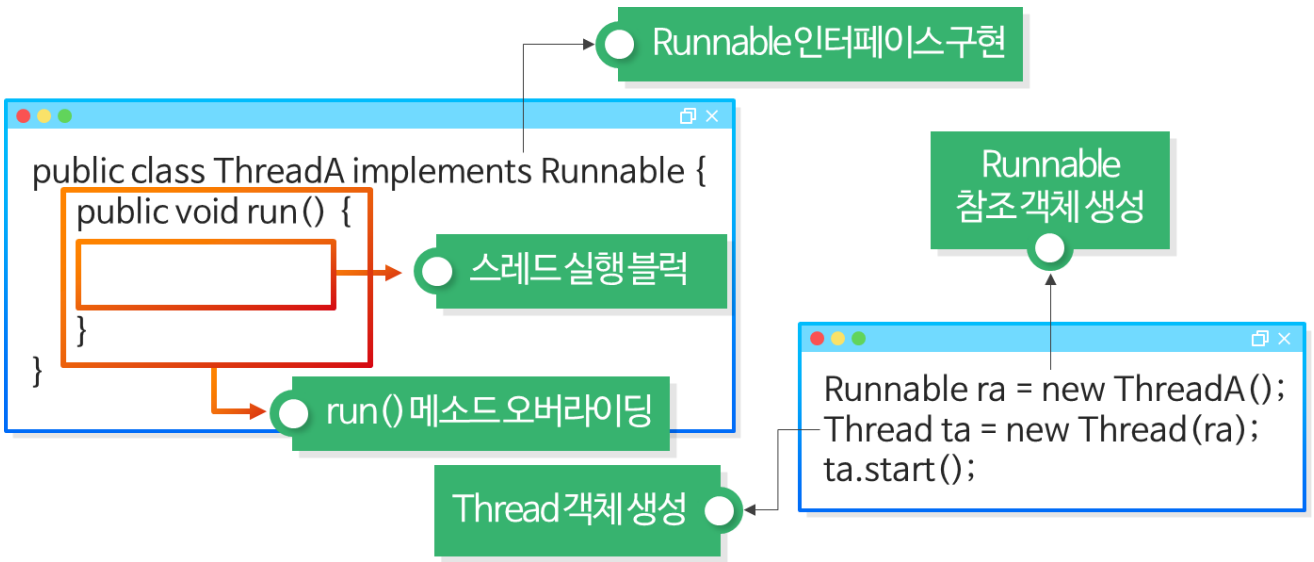
(1) Thread 클래스를 상속
(2) run() 메소드 오버라이딩
(3) Thread를 상속 받은 클래스의 객체 생성
(4) start() 메소드 호출



멀티 스레드 이해하기

4) Runnable 인터페이스 구현 방법

(1) Runnable 인터페이스를 구현
(2) run() 메소드 오버라이딩
(3) Runnable 인터페이스를 구현한 클래스의 객체 생성
(4) Runnable 구현 클래스의 객체를 이용하여 Thread 클래스의 객체 생성
(5) start() 메소드 호출



멀티 스레드 이해하기

멀티 스레드 제어

1) 스레드 우선순위 제어

(1) 우선순위에 따라 스레드가 실행될 수 있는 시간이 할당됨
(2) 우선순위 범위 : 1~10

- ✓ 숫자가 높을 수록 우선순위가 높음

(3) 기본 우선순위는 5
(4) 우선순위 관련 메소드 및 상수

주요 메소드 및 상수	설명
void setPriority(int newPriority)	현재 스레드의 우선순위 지정
int getPriority()	현재 스레드의 우선순위 반환
MAX_PRIORITY = 10 MIN_PRIORITY = 1 NORM_PRIORITY = 5	우선순위 상수

2) 데몬(daemon) 스레드

(1) 일반 스레드의 작업을 돕는 보조적인 스레드
(2) 일반 스레드가 모두 종료되면 자동 종료됨

- ✓ 일반 스레드는 종료 조건을 주어 동작
- ✓ 데몬 스레드는 종료 조건 없이 무한 반복 동작

(3) 데몬 스레드 관련 메소드

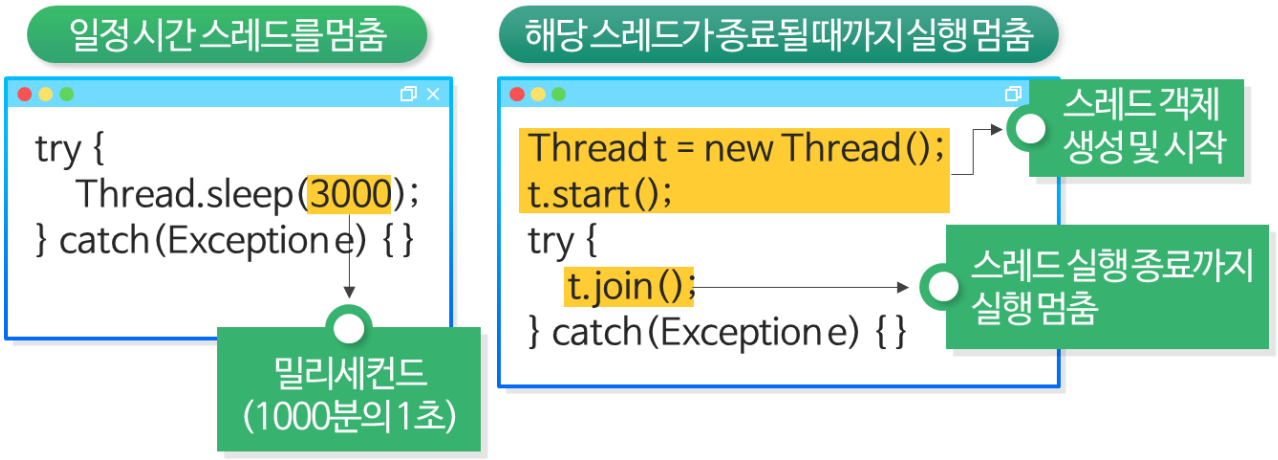
주요 메소드 및 상수	설명
void setDaemon(boolean on)	setDaemon(true) : 데몬 스레드로 설정
boolean isDaemon()	현재 스레드가 데몬 스레드인지 여부 반환

멀티 스레드 이해하기

3) 스레드 지연

(1) sleep() 메소드 : 일정시간 동안 스레드의 실행을 멈춤
(2) join() 메소드 : 해당 스레드의 동작이 종료될 때까지 실행을 멈춤

- try ~ catch 블록 내부에서 사용



4) 스레드 양보

(1) 스레드에게 주어진 실행시간을 다음 차례의 스레드에게 양보
(2) 하나의 스레드가 독점적으로 실행되지 않도록 함
(3) Thread.yield() 메소드 호출

실습

스레드 생성 및 제어 실습

실행 화면

```
[싱글 스레드 실행]:
main
01234
main
01234
[멀티 스레드 실행]
Thread-0:0
Thread-1:0
Thread-0:1
Thread-1:1
Thread-0:2
Thread-0:3
Thread-1:2 …
```

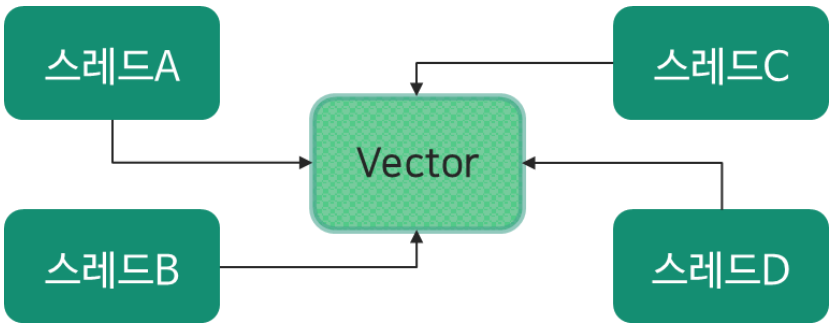
- 소스 파일명 : [ThreadExample.java]
- 자세한 내용은 실습 영상을 확인해보세요.

멀티 스레드 이해하기

멀티 스레드 동기화

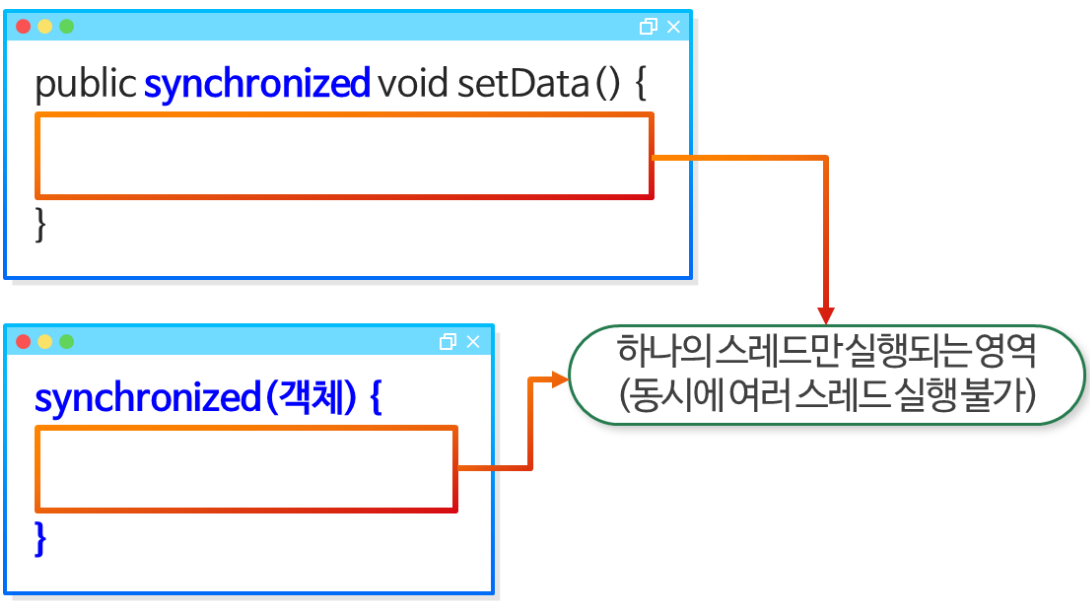
1) 동기화(synchronization)란?

- 여러 개의 스레드가 하나의 자원을 사용할 때 필요
- 하나의 스레드가 자원을 사용한 다음, 다른 스레드가 자원을 사용
- 하나의 스레드가 작업중인 것을 다른 스레드가 간섭하지 못하도록 함



2) 동기화 방법

(1) synchronized 키워드를 사용
(2) 메소드에 지정하거나 특정한 영역을 지정

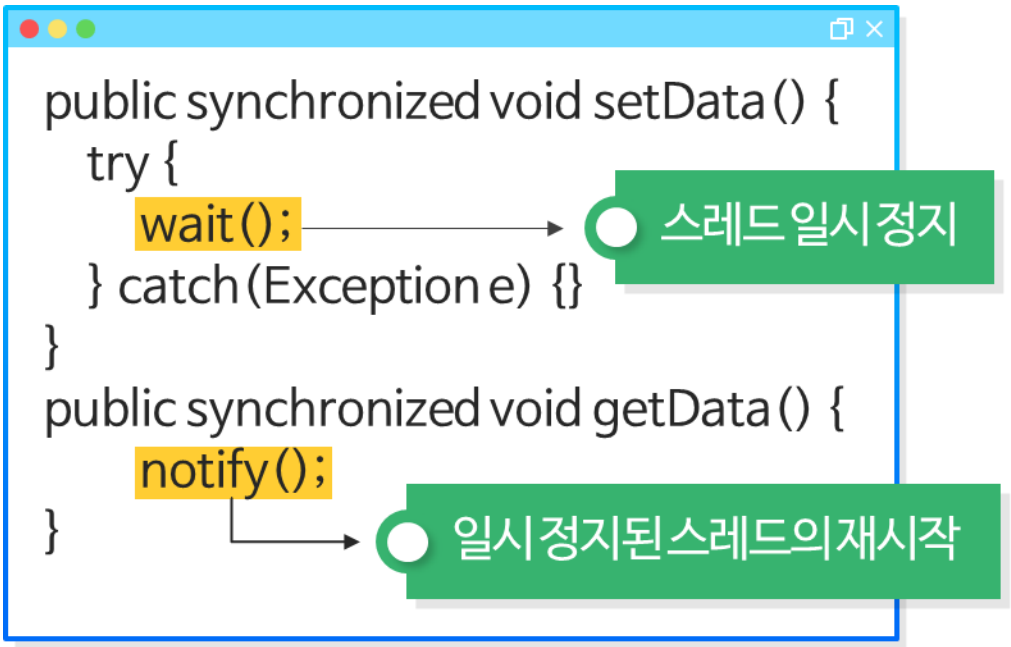


▸ 객체를 처리하는 특정 영역 지정

멀티 스레드 이해하기

3) 동기화 제어

(1) wait() 메소드 : 동기화 영역에서 스레드의 실행을 일시 정지시킴
(2) notify() 메소드 : 일시 정지된 스레드를 다시 실행하도록 함



실습

스레드 생성 및 제어 실습

실행 화면

```
[멀티 스레드 동기화 실습]
Thread-1: count increment : 11
Thread-0: count increment : 12
Thread-1: count increment : 13
Thread-0: count increment : 14
Thread-1: count increment : 15
Thread-0: count increment : 16
Thread-1: count increment : 17
Thread-0: count increment : 18
Thread-1: count increment : 19
Thread-0: count increment : 20
Thread-1: count increment : 21
Thread-0: count increment : 22…
```

- 소스 파일명 : [ThreadSync.java], [WaitNotify.java]
- 자세한 내용은 실습 영상을 확인해보세요.

정리하기

▪ 제네릭 이해하기

- 제네릭(generic)

  - 클래스나 메소드에서 처리할 자료형을 미리 지정하지 않음
  - 클래스의 객체를 생성할 때 자료형을 지정함
  - 주로 컬렉션 클래스와 인터페이스 및 반복자 클래스에 사용됨
  - 제네릭 사용의 장점
    - ➢ 하나의 클래스를 자료형 지정을 통하여 해당 자료형 전용 클래스로 사용
    - ➢ 객체 형 변환을 하지 않아도 되어 소스가 간결해짐
  - 제네릭 정의 방법
    - ➢ 꺽쇠표(<, >) 기호를 사용하여 영문 대문자 하나의 글자로 지정

정리하기

▪ 다차원 배열 이해하기

- 확장 for 문

  - 배열 또는 컬렉션을 위한 for문
  - 배열의 인덱스나 컬렉션의 반복자 없이도 반복 가능

- 열거형(enumeration type)

  - 상수들을 묶어서 정의한 자료형
  - 열거형 변수는 정의된 상수들만 지정할 수 있음
  - 열거형 사용
    - ➢ 클래스가 아니므로 객체 생성을 할 수 없음
    - ➢ 열거형 변수를 선언하여 사용

정리하기

▪ 배열 이해하기

- 멀티 스레드(multi-thread)의 예
  - 채팅 프로그램(상대방의 채팅내역이 화면에 표시되면서 동시에 내가 입력한 채팅내역을 전송)
  - 게임 프로그램(사용자의 동작도 수행하면서, 상대방의 동작도 동시에 수행)
- Thread 클래스 상속 방법
  - Thread 클래스를 상속
  - run() 메소드 오버라이딩
  - Thread를 상속받은 클래스의 객체 생성
  - start() 메소드 호출
- Runnable 인터페이스 구현 방법
  - Runnable 인터페이스를 구현
  - run() 메소드 오버라이딩
  - Runnable 인터페이스를 구현한 클래스의 객체 생성
  - Runnable 구현 클래스의 객체를 이용하여 Thread 클래스의 객체 생성
  - start() 메소드 호출
- 멀티 스레드 제어
  - 스레드 우선순위 제어 : 우선순위에 따라 스레드가 실행될 수 있는 시간이 할당됨
  - 데몬(daemon) 스레드 : 일반 스레드의 작업을 돕는 보조적인 스레드
  - 스레드 지연 : sleep() 메소드, join() 메소드
  - 스레드 양보 : Thread.yield() 메소드