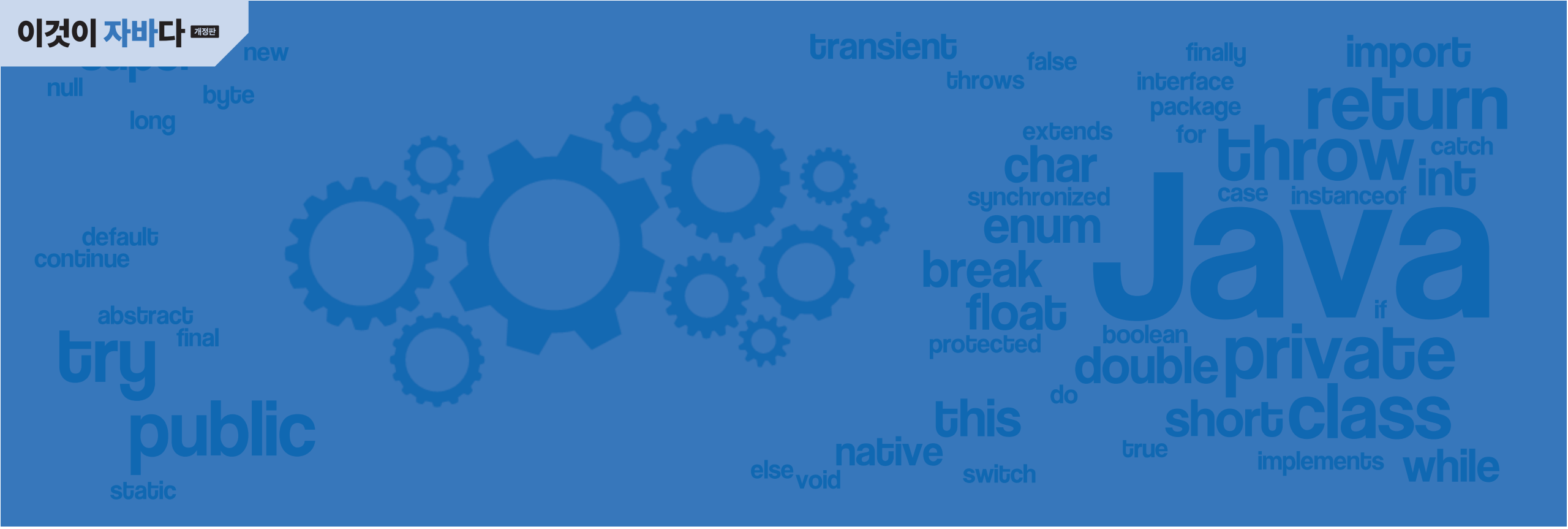
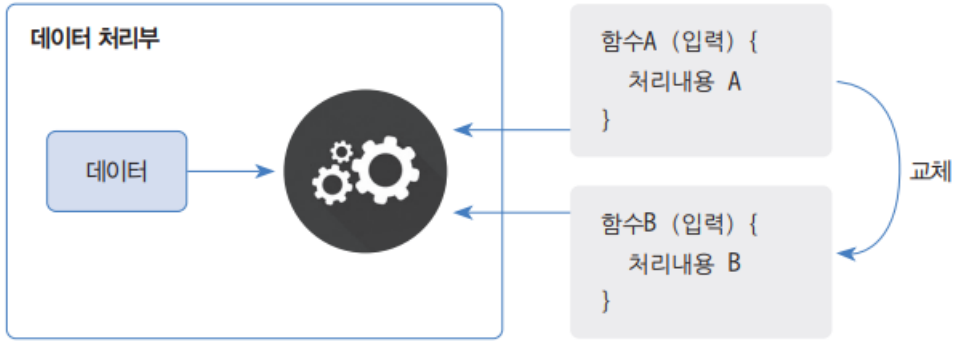


Chapter 16 람다식

▶ Contents

람다식

- 함수형 프로그래밍: 함수를 정의하고 이 함수를 데이터 처리부로 보내 데이터를 처리하는 기법
- 데이터 처리부는 제공된 함수의 입력값으로 데이터를 넣고 함수에 정의된 처리 내용을 실행
- 람다식: 데이터 처리부에 제공되는 함수 역할을 하는 매개변수를 가진 중괄호 블록이다.
- 자바는 람다식을 익명 구현 객체로 변환




```
람다식:  (매개변수, …) -> { 처리 내용 }
```

```
public interface Calculable {
  //추상 메소드
  void calculate(int x, int y);
}
```

```
action( (x, y) -> {
  int result = x + y;
  System.out.println(result);
});
```

함수형 인터페이스

- 인터페이스가 단 하나의 추상 메소드를 가지는 것

인터페이스

```
public interface Runnable {
  void run();
}
```

람다식

```
( ) -> { ··· }
```

인터페이스

```
@FunctionalInterface
public interface Calculable {
  void calculate(int x, int y);
}
```

람다식

```
( x, y ) -> { ··· }
```

- 인터페이스가 함수형 인터페이스임을 보장하기 위해서는 @FunctionalInterface 어노테이션을 붙임
- @FunctionalInterface: 컴파일 과정에서 추상 메소드가 하나인지 검사해 정확한 함수형 인터페이스를 작성할 수 있게 도와주는 역할

매개변수가 없는 람다식

- 함수형 인터페이스의 추상 메소드에 매개변수가 없을 경우 람다식 작성하기
- 실행문이 두 개 이상일 경우에는 중괄호를 생략할 수 없고, 하나일 경우에만 생략할 수 있음

```
( ) -> {
  실행문;
  실행문;
}
```

```
( ) -> 실행문
```

```
1   package ch16.sec02.exam02;
2
3   public class ButtonExample {
4     public static void main(String[] args) {
5       //Ok 버튼 객체 생성
6       Button btnOk = new Button();
7
8       //Ok 버튼 객체에 람다식(ClickListener 익명 구현 객체) 주입
9       btnOk.setClickListener(() -> {
10        System.out.println("Ok 버튼을 클릭했습니다.");
11      });
12
13      //Ok 버튼 클릭하기
14      btnOk.click();
15
16      //Cancel 버튼 객체 생성
17      Button btnCancel = new Button();
18
19      //Cancel 버튼 객체에 람다식(ClickListener 익명 구현 객체) 주입
20      btnCancel.setClickListener(() -> {
21        System.out.println("Cancel 버튼을 클릭했습니다.");
22      });
23
24      //Cancel 버튼 클릭하기
25      btnCancel.click();
26    }
27  }
```

매개값으로 람다식 대입

매개값으로 람다식 대입

매개변수가 있는 람다식

- 함수형 인터페이스의 추상 메소드에 매개변수가 있을 경우 람다식 작성하기
- 매개변수를 선언할 때 타입은 생략할 수 있고, 구체적인 타입 대신에 var를 사용할 수 있음

```
(타입 매개변수, … ) -> {
  실행문;
  실행문;
}
```

```
(var 매개변수, …) -> {
  실행문;
  실행문;
}
```

```
(매개변수, …) -> {
  실행문;
  실행문;
}
```

```
(타입 매개변수, … ) -> 실행문
```

```
(var 매개변수, …) -> 실행문
```

```
(매개변수, …) -> 실행문
```

- 매개변수가 하나일 경우에는 괄호를 생략 가능. 이때는 타입 또는 var를 붙일 수 없음

```
매개변수 -> {
  실행문;
  실행문;
}
```

```
매개변수 -> 실행문
```

리턴값이 있는 람다식

- 함수형 인터페이스의 추상 메소드에 리턴값이 있을 경우 람다식 작성하기
- return 문 하나만 있을 경우에는 중괄호와 함께 return 키워드를 생략 가능
- 리턴값은 연산식 또는 리턴값 있는 메소드 호출로 대체 가능

```
(매개변수, … ) -> {
  실행문;
  return 값;
}
```

```
(매개변수, …) -> return 값;
(매개변수, …) -> 값
```

메소드 참조

- 메소드를 참조해 매개변수의 정보 및 리턴 타입을 알아내 람다식에서 불필요한 매개변수를 제거

```
(left, right) -> Math.max(left, right);
```

정적 메소드와 인스턴스 메소드 참조

- 정적 메소드를 참조 시 클래스 이름 뒤에 :: 기호를 붙이고 정적 메소드 이름을 기술

```
클래스 :: 메소드
```

- 인스턴스 메소드일 경우에는 객체를 생성한 다음 참조 변수 뒤에 :: 기호를 붙이고 인스턴스 메소드 이름을 기술

```
참조변수 :: 메소드
```

매개변수의 메소드 참조

- 람다식에서 제공되는 a 매개변수의 메소드를 호출해서 b 매개변수를 매개값으로 사용

```
(a, b) -> { a.instanceMethod(b); }
```

- a의 클래스 이름 뒤에 :: 기호를 붙이고 메소드 이름을 기술

```
클래스 :: instanceMethod
```

```
1  package ch16.sec05.exam02;
2
3  public class MethodReferenceExample {
4    public static void main(String[] args) {
5      Person person = new Person();
6      person.ordering( String :: compareToIgnoreCase );
7    }
8  }
```

(a, b) -> a.compareToIgnoreCase(b)

메소드 참조

생성자 참조

- 객체를 생성하는 것. 람다식이 단순히 객체를 생성하고 리턴하도록 구성되면 람다식을 생성자 참조로 대치 가능

```
(a, b) -> { return new 클래스(a, b); }
```

- 클래스 이름 뒤에 :: 기호를 붙이고 new 연산자를 기술

```
클래스 :: new
```

- 생성자가 오버로딩되어 여러 개가 있을 경우, 컴파일러는 함수형 인터페이스의 추상 메소드와 동일한 매개변수 타입과 개수를 가지고 있는 생성자를 찾아 실행
- 해당 생성자가 존재하지 않으면 컴파일 오류 발생


transient
false
finally
import
throws
interface
new
byte
null
long
package
return
extends
for
throw
catch
char
int
synchronized
case
instanceof
enum
Java
default
continue
if
abstract
final
boolean
protected
try
double
private
do
this
short
class
public
native
true
implements
while
else
void
switch
static

Thank you!