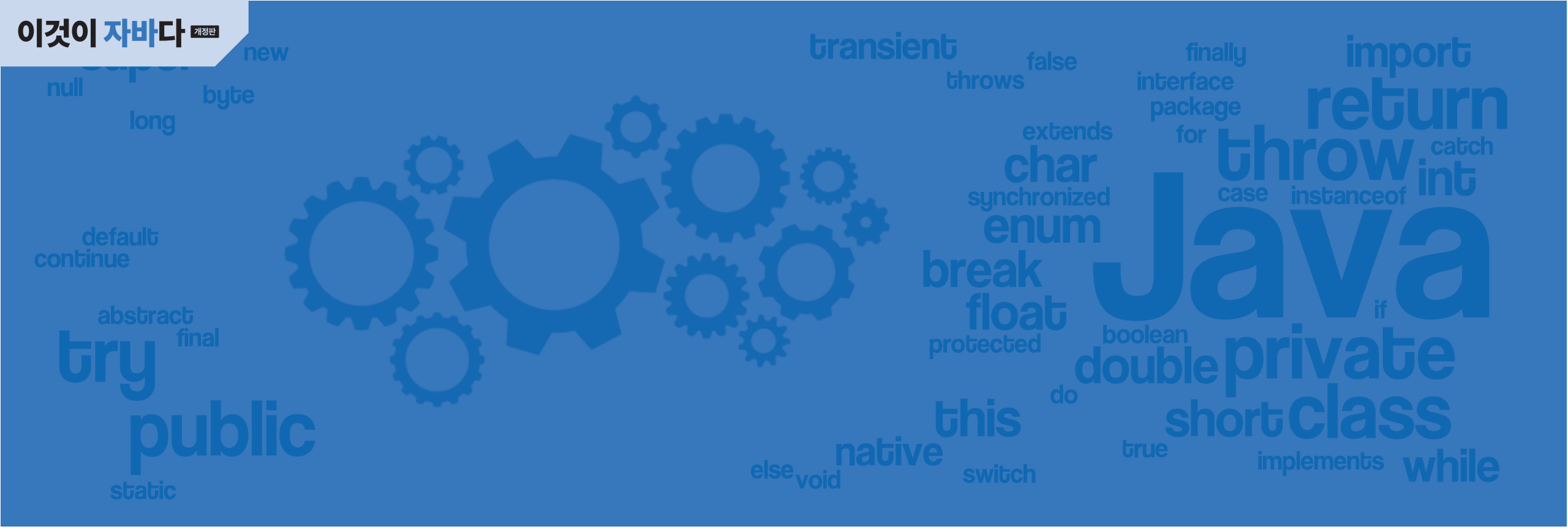


Chapter 13 제네릭

▶ Contents

제네릭

- 결정되지 않은 타입을 파라미터로 처리하고 실제 사용할 때 파라미터를 구체적인 타입으로 대체시키는 기능
- <T>는 T가 타입 파라미터임을 뜻하는 기호. 타입이 필요한 자리에 T를 사용할 수 있음을 알려줌

```
public class Box <T> {
  public T content;
}
```

```
Box<String> box = new Box<String>();
```

→

```
Box<String> box = new Box<>();
```

```
Box<Integer> box = new Box<Integer>();
```

→

```
Box<Integer> box = new Box<>();
```

제네릭 타입

- 결정되지 않은 타입을 파라미터로 가지는 클래스와 인터페이스
- 선언부에 '< >' 부호가 붙고 그 사이에 타입 파라미터들이 위치

```
public class 클래스명<A, B, ···> { ... }
public interface 인터페이스명<A, B, ···> { ... }
```

- 타입 파라미터는 일반적으로 대문자 알파벳 한 글자로 표현
- 외부에서 제네릭 타입을 사용하려면 타입 파라미터에 구체적인 타입을 지정. 지정하지 않으면 Object 타입이 암묵적으로 사용

제네릭 메소드

- 타입 피라미터를 가지고 있는 메소드. 타입 파라미터가 메소드 선언부에 정의
- 리턴 타입 앞에 < > 기호 추가하고 타입 파라미터 정의 후 리턴 타입과 매개변수 타입에서 사용

```
public <A, B, ···> 리턴타입 메소드명(매개변수, ...) { ... }
```

타입 파라미터 정의

- 타입 파라미터 T는 매개값의 타입에 따라 컴파일 과정에서 구체적인 타입으로 대체

```
public <T> Box<T> boxing(T t) { ··· }
```

```
① Box<Integer> box1 = boxing(100);
② Box<String> box2 = boxing("안녕하세요");
```

제한된 타입 파라미터

- 모든 타입으로 대체할 수 없고, 특정 타입과 자식 또는 구현 관계에 있는 타입만 대체할 수 있는 타입 파라미터

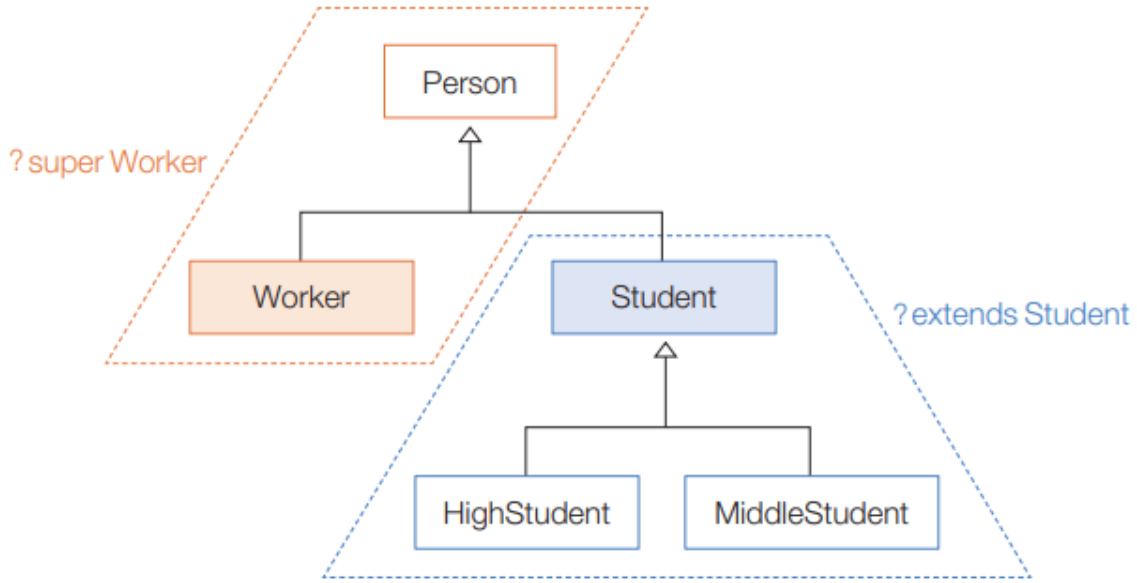
```
public <T extends 상위타입> 리턴타입 메소드(매개변수, ...) { ... }
```

- 상위 타입은 클래스뿐만 아니라 인터페이스도 가능

```
public <T extends Number> boolean compare(T t1, T t2) {
  double v1 = t1.doubleValue();   //Number의 doubleValue() 메소드 사용
  double v2 = t2.doubleValue();   //Number의 doubleValue() 메소드 사용
  return (v1 == v2);
}
```

와일드카드 타입 파라미터

- 제네릭 타입을 매개값이나 리턴 타입으로 사용할 때 범위에 있는 모든 타입으로 대체할 수 있는 타입 파라미터. ?로 표시




```
리턴타입 메소드명(제네릭타입<? extends Student> 변수) { … }
```

```
리턴타입 메소드명(제네릭타입<? super Worker> 변수) { … }
```

```
리턴타입 메소드명(제네릭타입<?> 변수) { … }
```


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