

ITEM 26

로 타입은 사용하지 말라

CONTENTS

- | | |
|-----------------|---------------|
| 1. 제네릭이란 | 4. 로 타입 대체 1 |
| 2. 로 타입 만든 이유 | 5. 로 타입 대체 2 |
| 3. 로 타입 사용 X 이유 | 6. 로 타입 사용 예외 |

CONTENTS

1. 제네릭이란

4. 로 타입 대체 1

2. 로 타입 만든 이유

5. 로 타입 대체 2

3. 로 타입 사용 X 이유

6. 로 타입 사용 예외

제네릭이란?

클래스 or 인터페이스 선언에 **타입 매개변수**

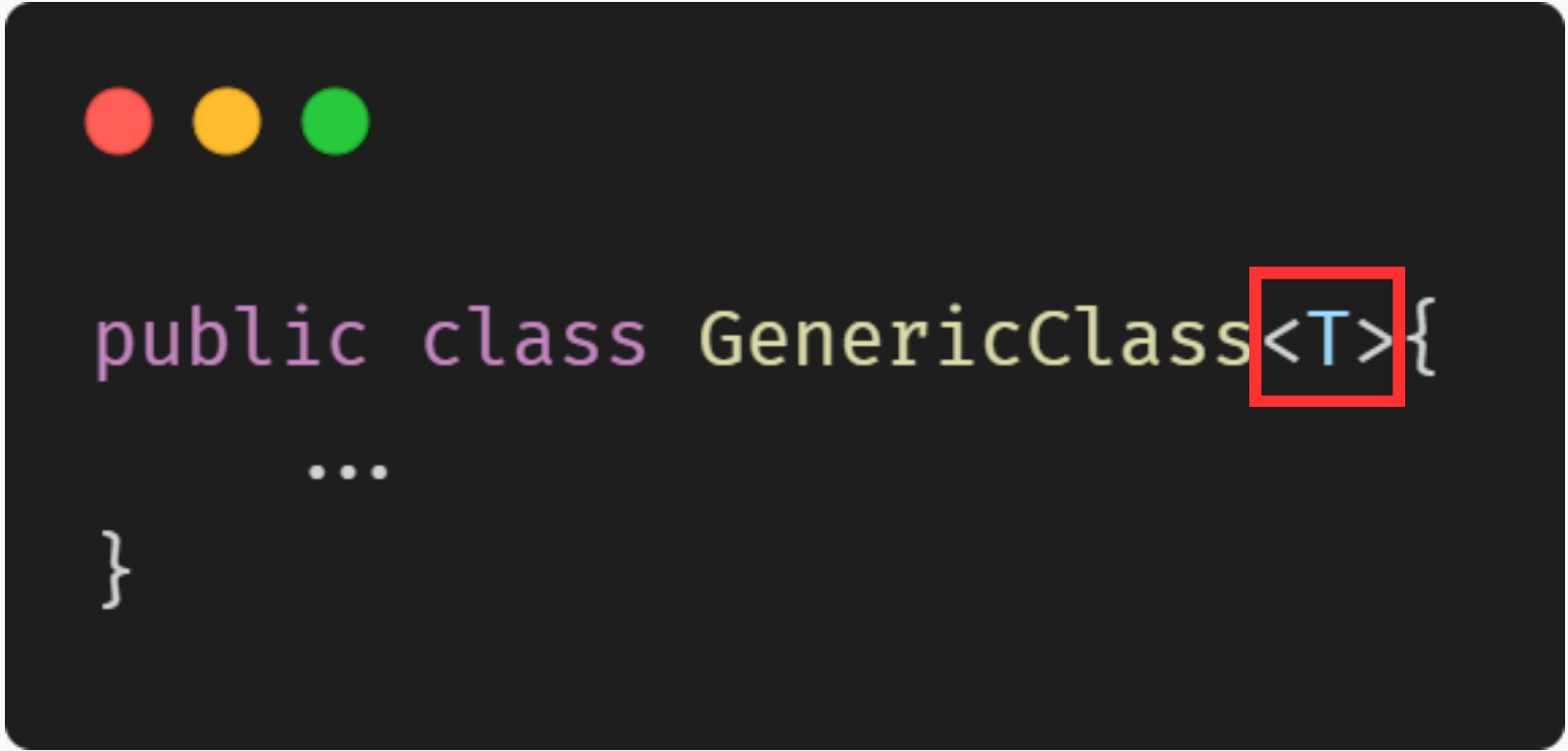


제네릭 클래스 or 제네릭 인터페이스 = **제네릭 타입**

제네릭이란?

타입 매개변수

클래스 or 인터페이스 선언에 **타입 매개변수**



```
public class GenericClass<T>{  
    ...  
}
```

타입 매개 변수 = **타입을 변수로 표시**

제네릭이란?

제네릭 클래스 or 제네릭 인터페이스
= 제네릭 타입



제네릭 = 타입 일반화,
타입 외부에서 지정



제네릭이란?

매개변수화 타입

제네릭 타입은 매개변수화 타입 정의



```
List<String> nameList;
```

List<String> 타입 = 매개변수화 타입

제네릭이란?

매개변수화 타입

List<T> = 제네릭 타입

List<String> 타입 = 매개변수화 타입

T 타입 = 형식 타입 매개변수

String = 실제 타입 매개변수

제네릭이란? 로 타입

제네릭 타입 정의 시 **로 타입도 함께 정의됨**

제네릭 타입에서 **타입 매개변수 사용 X**



제네릭이란?
로 타입

List<T> = 제네릭 타입

T 타입 = 형식 타입 매개변수

List = 로 타입

제네릭이란?

매개변수화 타입

- 제네릭 타입

- 매개변수화 타입

- 형식 타입 매개변수
 - 실제 타입 매개변수

- 로 타입

- **List<T>**

- **List<String>**

- **T**

- **String**

- **List**

CONTENTS

1. 제네릭이란

4. 로 타입 대체 1

2. 로 타입 만든 이유

5. 로 타입 대체 2

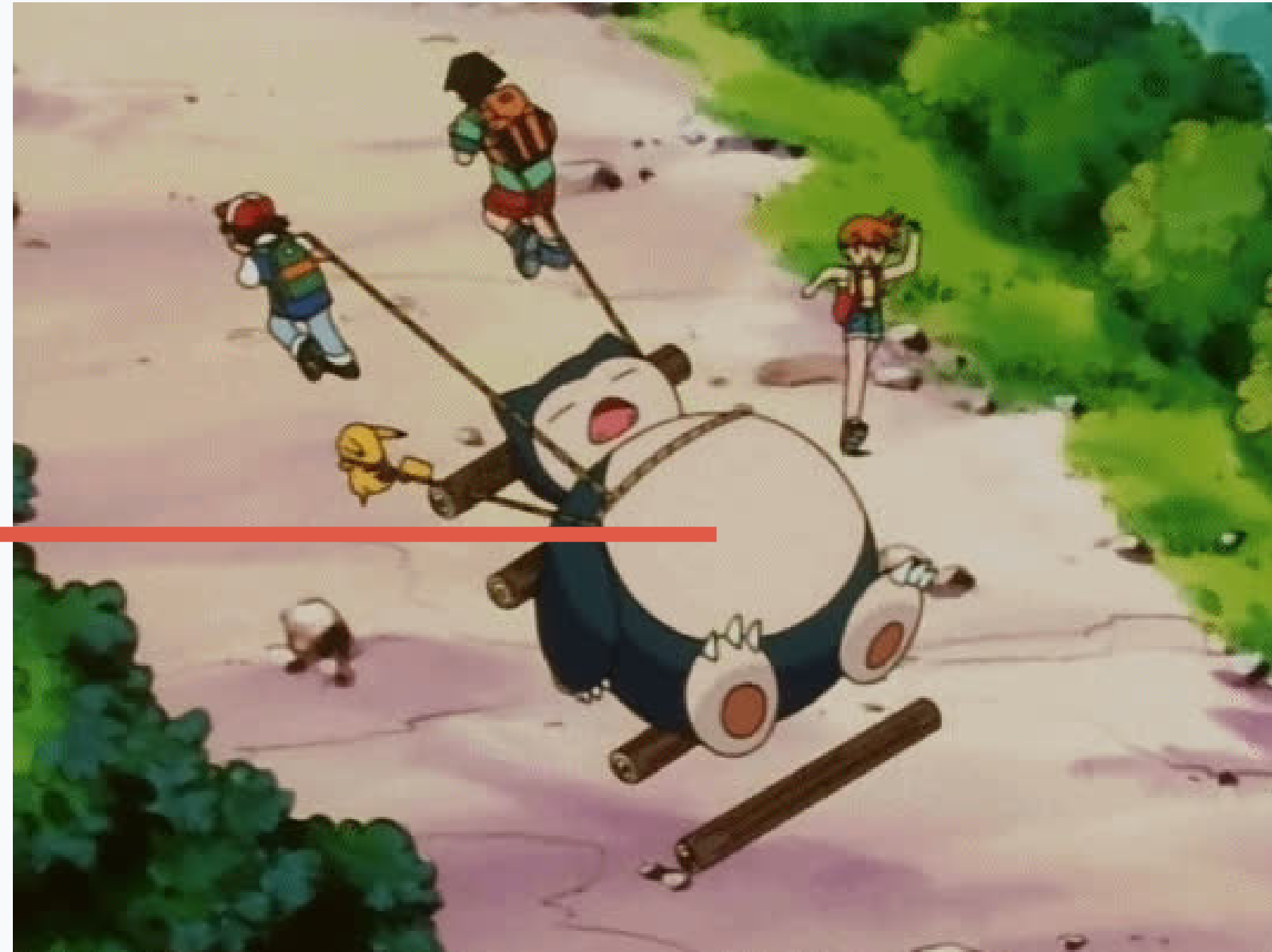
3. 로 타입 사용 X 이유

6. 로 타입 사용 예외

로 타입 만든 이유

제네릭 이전 코드와의 **호환성**

제네릭
이전 코드



로 타입 만든 이유

제네릭 이전 코드

List



제네릭 이후 코드

List<String>

로 타입 만든 이유

```
public class MainClass {  
  
    public void rawTypeParameter(GenericClass parameter){  
        parameter.sayHello();  
    }  
  
    public void parameterizedTypeParameter(GenericClass<String> parameter){  
        parameter.sayHello();  
    }  
  
    public static void main(String[] args) {  
        GenericClass<String> temp1 = new GenericClass<>();  
        GenericClass temp2 = new GenericClass();  
  
        MainClass mainClass = new MainClass();  
  
        mainClass.parameterizedTypeParameter(temp2);  
        mainClass.rawTypeParameter(temp1);  
    }  
}
```

CONTENTS

1. 제네릭이란

4. 로 타입 대체 1

2. 로 타입 만든 이유

5. 로 타입 대체 2

3. 로 타입 사용 X 이유

6. 로 타입 사용 예외

로 타입 사용 **X** 이유

제네릭이 안겨주는 **안전성, 표현력** 모두 잃음



로 타입 사용 X 이유

제네릭 쓰면 다른 타입 인스턴스 막아주는 안전성 있다.

제네릭 쓰면 특정 타입만 사용하겠다고 의도 표현하는 표현력 있다.

로 타입 사용 ✕ 이유

로 타입 사용 시 문제

```
// Collection에서 Stamp 인스턴스만 취급한다.  
private final Collection stamps = ... ;  
  
stamps.add(new Stamp( ... ));  
  
// 실수로 Coin 인스턴스 넣는다.  
stamps.add(new Coin( ... ));  
  
// Coin 인스턴스 꺼내면 ClassCastException 발생  
Stamp stamp = (Stamp) stamps.get( ... );
```

아무 타입 추가 가능

다른 타입 꺼내 형 변환 오류 런타임에 발생

로 타입 사용 **X** 이유

제네릭 사용 후 해결



```
private final Collection<Stamp> stamps = ... ;
```

```
stamps.add(new Stamp( ... ));
```

```
// 실수로 Coin 인스턴스 넣기 불가능
```

```
stamps.add(new Coin( ... )); ← 컴파일 오류 발생
```

```
// 자동 형변환 해주며 형변환 실패 X
```

```
Stamp stamp = stamps.get( ... );
```

다른 타입 인스턴스 추가 시 **컴파일 오류**
자동 형 변환 해줌

CONTENTS

1. 제네릭이란

4. 로 타입 대체 1

2. 로 타입 만든 이유

5. 로 타입 대체 2

3. 로 타입 사용 X 이유

6. 로 타입 사용 예외

로 타입 대체 1

로 타입
List



매개변수화 타입
List<Object>



로 타입 대체 1

List<Object>



```
private final Collection<Object> stamps1 = ... ;
```

```
private final Collection stamps2 = ... ;
```

```
stamps1.add(new Stamp( ... ));  
stamps1.add(new Coin( ... ));
```

```
stamps2.add(new Stamp( ... ));  
stamps2.add(new Coin( ... ));
```

둘 다 모든 타입 인스턴스 추가 가능

로 타입 대체 1

List<Object>

```
public void genericMethod(List<Object> temp){~}

public void rawTypeMethod(List temp){~}

public static void main{
    List<String> temp = new ArrayList<>();

    genericMethod(temp); ← 컴파일 오류 발생

    rawTypeMethod(temp);
}
```

메서드 파라미터로 사용 시 로 타입보다 **타입 안전성** 얻는다.

로 타입 대체 1

List<Object>

Why??

List == List<String> 상위 타입

List<Object> != List<String> 상위 타입

CONTENTS

1. 제네릭이란

4. 로 타입 대체 1

2. 로 타입 만든 이유

5. 로 타입 대체 2

3. 로 타입 사용 X 이유

6. 로 타입 사용 예외

로 타입 대체 2

로 타입
List



비한정적 와일드카드 타입
List<?>



로 타입 대체 2

List<?>

비한정적 와일드카드 타입

<?>로 표현됨

아직 알 수 없는 타입 의미

모든 종류 객체 다룰 수 있음을 의미

로 타입 대체 2

List<?>

```
public void genericMethod(List<?> temp){~}

public void rawTypeMethod(List temp){~}

public static void main{
    List<String> temp1 = new ArrayList<>();
    genericMethod(temp1);
    rawTypeMethod(temp1);

    List<Integer> temp2 = new ArrayList<>();
    genericMethod(temp2);
    rawTypeMethod(temp2);
}
```

둘 다 메서드 파라미터로 모든 제네릭 타입 가능

로 타입 대체 2

List<?>

```
public void genericMethod(List<?> temp){
    temp.add(new Integer( ... ));
    teamp.add(null);
}

public void rawTypeMethod(List temp){
    temp.add(new Integer( ... ));
    temp.add(new Double( ... ));
    teamp.add(null);
}
```

← 컴파일 오류 발생

비한정적 와일드카드 타입은 **null만 추가 가능**

로 타입 대체 2

List<?>

List = 모든 타입 원소 추가 가능

검사와
오류 발생

List<Object> = null만 추가 가능

타입 불변식 훼손 막아줌

값을 꺼내서 형 변환, 사용 시 안전

CONTENTS

- | | |
|-----------------|---------------|
| 1. 제네릭이란 | 4. 로 타입 대체 1 |
| 2. 로 타입 만든 이유 | 5. 로 타입 대체 2 |
| 3. 로 타입 사용 X 이유 | 6. 로 타입 사용 예외 |

로 타입 사용 예외

1. class 리터럴

2. instanceof 연산자



로 타입 사용 예외

1. **class** 리터럴

String 리터럴 = “hello”

class 리터럴 = String.class

class 리터럴은 특정 클래스의 Class 객체 얻기 위한 것

로 타입 사용 예외

1. **class** 리터럴

class 리터럴에 매개변수화 타입 사용 불가

List.class, Integer.class = ○

List<String>.class, List<?>.class = ✗

로 타입 사용 예외

2. Instanceof 연산자

런타임에는 제네릭 타입 **정보가 지워짐**

로 타입, 매개변수화 타입 **똑같이 동작**



제너릭의 장점 지우는

로 타입은 사용하지 말자