퀴즈- 제너릭

1. 제네릭에 대한 설명으로 틀린 것은 무엇입니까?

❶ 컴파일 시 강한 타입 체크를 할 수 있다.

❷ 타입 변환(casting)을 제거한다.

❸ 제네릭 타입은 타입 파라미터를 가지는 제네릭 클래스와 인터페이스를 말한다.

❹ 제네릭 메소드는 리턴 타입으로 타입 파라미터를 가질 수 없다.

2. ContainerExample 클래스의 main() 메소드는 Container 제네릭 타입을 사용하고 있습니다. main() 메소드에서 사용하는 방법을 참고해서 Container 제네릭 타입을 선언해보세요.

```
public class ContainerExample {
  public static void main(String[] args) {
    Container<String> container1 = new Container<String>();
    container1.set("홍길동");
    String str = container1.get();

    Container<Integer> container2 = new Container<Integer>();
    container2.set(6);
    int value = container2.get();
  }
}
```

3. ContainerExample 클래스의 main() 메소드는 Container 제네릭 타입을 사용하고 있습니다. main() 메소드에서 사용하는 방법을 참고해서 Container 제네릭 타입을 선언해보세요.

```
public class ContainerExample {
  public static void main(String[] args) {
    Container<String, String> container1 = new Container<String, String>();
    container1.set("홍길동", "도적");
    String name1 = container1.getKey();
    String job = container1.getValue();

    Container<String, Integer> container2 = new Container<String, Integer>();
    container2.set("홍길동", 35);
    String name2 = container2.getKey();
    int age = container2.getValue();
  }
}
```

4. 다음 Util 클래스의 정적 getValue() 메소드는 첫 번째 매개값으로 Pair 타입과 하위 타입만 받고, 두 번째 매개값으로 키값을 받습니다. 리턴값은 키값이 일치할 경우 Pair에 저장된 값을 리턴하고, 일치하지 않으면 null을 리턴하도록 Util 클래스와 getValue() 제네릭 메소드를 작성해보세요.

```
public class UtilExample {
  public static void main(String[] args) {
    Pair<String, Integer> pair = new Pair<>( "홍길동" , 35 );
    Integer age = Util.getValue(pair, "홍길동" );
    System.out.println(age);                                  일치

    ChildPair<String, Integer> childPair = new ChildPair<>( "홍삼원" , 20 );
    Integer childAge = Util.getValue(childPair, "홍삼순" );
    System.out.println(childAge);                                  불일치

    /*OtherPair<String, Integer> otherPair = new OtherPair<>("홍삼원", 20);
    //OtherPair는 Pair를 상속하지 않으므로 컴파일 에러가 발생
    int otherAge = Util.getValue(otherPair, "홍삼원");
    System.out.println(otherAge);*/
  }
}
```

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

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

  public K getKey()   { return key; }
  public V getValue() { return value; }
}
```

```
public class ChildPair<K, V> extends Pair<K,V> {
  public ChildPair(K k, V v) {
    super(k, v);
  }
}
```

```
public class OtherPair<K, V> {
  private K key;
  private V value;

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

  public K getKey()   { return key; }
  public V getValue() { return value; }
}
```