

# 클래스, 객체

익셉션, try-catch, checked vs unchecked 익셉션, Wrapper 클래스, 박싱, 언박싱, 제네릭 클래스, LinkedList, HashMap

# 익셉션(Exception)

## ✚ 익셉션(예외, exception)

- 프로그램 실행 도중에 정상적인 실행을 제한하는 예외 상황을 지칭하는 용어

## ✚ 익셉션 발생 시 적절한 처리가 요구됨

```
public class Test {  
    public static void main(String[] args) {  
        int      n1=0, n2=5;  
        System.out.println(n2/n1);  
    }  
}
```

divide by zero

<실행결과>

Exception in thread "main"

java.lang.ArithmeticException: / by zero

```
public class Test {  
    public static void main(String[] args) {  
        User user=new User("gdhong", "홍길동", "1234");  
    }  
}
```

← 비밀번호

- User 객체 생성 규정 → User 객체의 비밀번호 길이는 **최소 6글자 이상** 되어야 함
- "1234"는 위의 비밀번호 규정을 만족하지 못하므로 User 객체는 생성되면 안 되므로 이러한 예외 상황에 대한 처리가 필요함

# 익셉션 처리: try-catch 절 사용

## 익셉션 처리 미적용

```
public class User {
    String id;
    String name;
    private String password;
    public User(String id, String name, String password) {
        this.id=id;
        this.name=name;
        this.password=password;
    }
    @Override
    public String toString(){
        return id+","+name;
    }
}

public class Test {
    public static void main(String[] args) {
        User user=new User("gdhong", "홍길동", "1234");
        System.out.println(user);
    }
}
```

## 익셉션 처리 적용

```
public class User {
    String id;
    String name;
    private String password;
    public User(String id, String name, String password) throws Exception {
        if(password.length()<6) throw new Exception("패스워드는 6 문자 이상");
        this.id=id;
        this.name=name;
        this.password=password;
    }
    @Override
    public String toString() {
        return id+","+name;
    }
}

public class Test {
    public static void main(String[] args) {
        try {
            User user= new User("gdhong", "홍길동", "1234");
            System.out.println(user);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Exception이 발생할 수 있는 메소드라는 표시

예외 상황 감지 및 익셉션 발생 시킴

Exception이 발생할 수 있는 메소드를 호출하는 경우 반드시 try {} catch {} 절로 감싸야 한다

# 익셉션 처리 예: FileInputStream

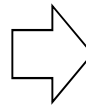
```
public class FileInputStream extends InputStream {  
    public FileInputStream(String name) throws FileNotFoundException {  
        ...  
    }  
    ...  
}
```

↑  
Exception이 발생할 수 있는  
메소드라는 표시

Exception  
| extends  
IOException  
| extends  
FileNotFoundException

```
public class Test {  
    public static void main(String[] args) {  
        FileInputStream fis=new FileInputStream("C:/A.txt");  
    }  
}
```

- Exception이 발생할 수 있는 메소드는 try {} catch {}로 감싸지 않으면 오류 발생
- 디스크에 C:/A.txt 파일이 존재하지 않는 경우 이후의 처리가 무의미해지므로 FileNotFoundException이라는 예외 발생될 수 있음



```
public class Test {  
    public static void main(String[] args) {  
        try {  
            FileInputStream fis=new FileInputStream("C:/A.txt");  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

←  
Exception이 발생할 수 있는 메소드를  
try {} catch {}로 감싸 예외 처리 적용

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            FileInputStream fis=new FileInputStream("C:/A.txt");  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

# 익셉션 처리 실습

✚ 클래스 User를 다음 조건에 따라 작성하시오

- 필드 email(메일주소, 문자열)
  - ◆ 참고: Boolean String.contains(CharSequence)
- email을 파라미터로 전달 받는 생성자
  - ◆ 파라미터로 전달받은 email 내 @ 누락 시 익셉션 메시지 “@ 누락” 출력
- email을 반환하는 toString()

CharSequence  
| implements  
String

✚ 클래스 Test의 main() 내 다음 절차를 코딩하시오

- User 객체(메일주소 gdhong@ks.ac.kr) 생성 시도
  - ◆ 생성 성공 시 객체 출력
  - ◆ 실패 시 익셉션 메시지 출력
- User 객체(메일주소 yhlee) 생성 시도
  - ◆ 생성 성공 시 객체 출력
  - ◆ 실패 시 익셉션 메시지 출력

# Checked vs. Unchecked Exception

## ✚ Checked 익셉션

- 컴파일 시점에 check(검사)되는 익셉션
- Checked 익셉션을 발생시키는 메소드를 호출하는 경우
  - ◆ try-catch 절이나 throws 절을 사용하여 checked 익셉션 처리해야 함

## ✚ Unchecked 익셉션

- 컴파일 시점에 check(검사)되지 않는 익셉션
  - ◆ 예) ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException, NumberFormatException

# Checked 익셉션: 두 가지 처리 방법

## Checked 익셉션

- 컴파일 시점에 check(검사)되는 익셉션
- Checked 익셉션을 발생시키는 메소드를 호출하는 경우
  - ◆ 해당 메소드를 try 절에 넣고 익셉션 처리 코드를 catch 절에 넣는 방식으로 try-catch 절을 사용해야 함 (아래 우측 상단 코드)
  - ◆ 혹은 해당 메소드를 호출하는 모메소드 선언부에 throws 절을 추가하여 발생하는 익셉션을 모메소드를 호출하는 코드로 다시 throw 하도록 해야 함 (아래 우측 하단 코드)

```
public class Test {  
    public static void main(String[] args) {  
        FileInputStream fis=new FileInputStream("C:/A.txt");  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        try {  
            FileInputStream fis=new FileInputStream("C:/A.txt");  
        } catch (FileNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) throws FileNotFoundException {  
        FileInputStream fis=new FileInputStream("C:/A.txt");  
    }  
}
```

# 자바 API 문서

✚ 다음 사이트에서 자바 API 문서를 확인할 수 있다

● <http://docs.oracle.com/javase/7/docs/api/>

✚ String.split() 메소드 설명

● <http://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

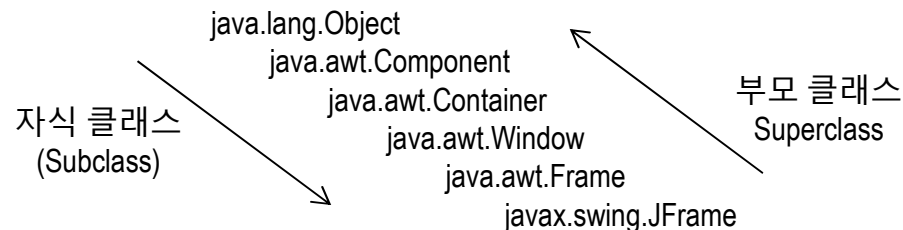
```
public String[] split(String regex)
```

↑  
반환 값 자료형

↑  
입력 파라미터 자료형

✚ JFrame의 클래스 계층도

● <http://docs.oracle.com/javase/7/docs/api/javax/swing/JFrame.html>



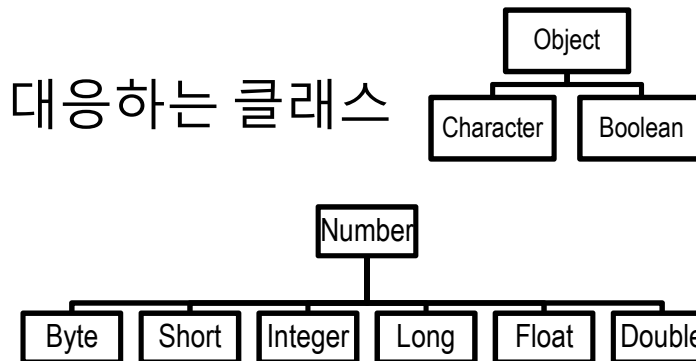


# Wrapper 클래스

## Wrapper 클래스

- 기본 자료형(primitive data type)에 대응하는 클래스

기본 자료형	Wrapper 클래스
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean



```
public class Test {  
    public static void main(String[] args) {  
        Integer n=new Integer(95);  
        String v="점 수="+n.toString();  
        System.out.println(v);  
  
        int score=3+n.intValue();  
        System.out.println(score);  
    }  
}
```

# Autoboxing, unboxing (박싱, 언박싱)

## Boxing (박싱)

- 기본 자료형 값을 대응하는 Wrapper 클래스 객체로 변환하는 작업

## Unboxing (언박싱)

- Wrapper 클래스 객체를 기본 자료형 값으로 변환하는 작업

```
public class Test {  
    public static void main(String[] args) {  
        // autoboxing  
        Integer n=95; // Integer n=new Integer(100);  
  
        // unboxing  
        int m=n; // int m=n.intValue();  
  
        System.out.println(n); // println(Object x)  
        System.out.println(m); // println(int x)  
    }  
}
```

## 실습

- Autoboxing을 이용하여 3.14를 Double형 참조 변수 pi에 저장하는 코드를 작성하시오
- 변수 pi에 저장된 값을 double 변수 x에 저장하는 코드를 작성하시오

## 실습

- Autoboxing을 이용하여 1~10까지의 정수를 Integer 배열 n에 저장하는 코드를 작성하시오

# Generic class (제네릭 클래스)

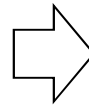
```
public class Value {  
    Object v;  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Value value=new Value();  
        value.v=new Integer(95);  
        String s=(String) value.v; // compile OK, runtime Error  
        System.out.println(s);  
    }  
}
```

value 객체에 저장된 값이 문자열이라고 생각하고 코딩  
컴파일은 문제 없으나 실행 시 오류(ClassCastException) 발생  
이러한 유형의 오류를 컴파일 시점에 확인할 수 있는 장치 필요

# Generic class (제네릭 클래스)

```
public class Value {  
    Object v;  
}
```

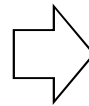


제네릭 클래스

```
public class Value<T> {  
    T v;  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Value value=new Value();  
        value.v=new Integer(95);  
        String s=(String) value.v; // runtime Error  
        System.out.println(s);  
    }  
}
```

실행 시점에 오류 발생  
컴파일 시 오류 확인 불가



```
public class Test {  
    public static void main(String[] args) {  
        Value<Integer> value=new Value<Integer>();  
        value.v=new Integer(95);  
        String s=(String) value.v; // compile Error  
        System.out.println(s);  
    }  
}
```

컴파일 시 오류 확인

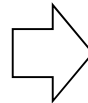
```
Value<int>value=new Value<int>(); // 오류
```

- 제네릭 클래스의 타입 파라미터에는 non-primitive 자료형을 전달해야 함
- int, double 등 기본 자료형은 사용 불가 (기본 자료형 대신 Wrapper 클래스 사용)

참 조: <https://docs.oracle.com/javase/tutorial/java/generics/types.html>

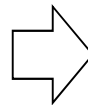
# Generic class (제네릭 클래스)

```
public class Player {  
    int record;  
    public void setRecord(int record) {  
        this.record = record;  
    }  
    public int getRecord() {  
        return record;  
    }  
}
```



```
public class Player<T> {  
    T record;  
    public void setRecord(T record) {  
        this.record = record;  
    }  
    public T getRecord() {  
        return record;  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
  
        Player p1=new Player();  
        p1.setRecord(32); // 32점  
        System.out.println(p1.getRecord());  
  
        Player p2=new Player();  
        p2.setRecord(9.875); // 9.875초  
    }  
}
```



```
public class Test {  
    public static void main(String[] args) {  
        Player<Integer> p1=new Player<Integer>();  
        p1.setRecord(32); // 단일 경기 최다 점수 기록 32점  
        System.out.println(p1.getRecord());  
        Player<Double> p2=new Player<Double>();  
        p2.setRecord(9.875); // 100미터 기록 9.875초  
        System.out.println(p2.getRecord());  
        Player<Integer []> p3=new Player<Integer []>();  
        Integer v[]={210, 195, 220}; // 높이뛰기 기록  
        p3.setRecord(v);  
        Integer w[]=p3.getRecord();  
        for (int i = 0; i < w.length; i++) System.out.println(w[i]);  
    }  
}
```

현재 Player 클래스를 통해서는 정수, 실수, 정수 배열 등 다양한 유형의 기록 자료들을 저장하기 어려움

오류

# JDK 제네릭 클래스 사용 예: LinkedList

```
public class Test {  
    public static void main(String[] args) {  
        LinkedList<String> country=new LinkedList<>();  
        country.add("한국");  
        country.add("미국");  
        country.add("일본");  
        System.out.println(country);  
    }  
}
```

## 실습

- 다음 점수 값들의 리스트 [85, 91, 73]를 LinkedList 객체로 생성 후 출력하시오.
- 요일 문자들의 리스트 [월, 화, 수, 목, 금, 토, 일]를 LinkedList 객체로 생성 후 출력하시오.
- 다음 기록 값들의 리스트 [12.9, 9.98, 10.52]를 LinkedList 객체로 생성 후 출력하시오.

## 실습

- 클래스 User는 email(메일주소, 문자열) 필드, name(이름, 문자열) 필드, name과 email을 파라미터로 전달 받는 생성자, name과 email 값을 반환하는 toString()으로 구성된다. 클래스 User를 작성하시오.
- yhlee@ee.ks.ac.kr(이영희), gdhong@ie.ks.ac.kr(홍길동), yhpark@cs.ks.ac.kr(박영희)에 해당하는 User 객체들의 목록을 LinkedList 객체로 생성 후 출력하시오.

```
public class Student {  
    String id;  
    int score;  
    public Student(String id, int score) {  
        this.id=id;  
        this.score=score;  
    }  
    @Override  
    public String toString() {  
        return "<" + id + ", " + score + ">";  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s=new Student("s-001", 98);  
        LinkedList<Student> list=new LinkedList<>();  
        list.add(s);  
        list.add(new Student("s-002", 100));  
        System.out.println(list);  
    }  
}
```

# JDK 제네릭 클래스 사용 예: HashMap

```
public class Test {  
    public static void main(String[] args) {  
        HashMap<String, Integer> rank=new HashMap<>();  
        rank.put("한국", 7);  
        rank.put("중국", 10);  
        rank.put("일본", 25);  
        System.out.println(rank);  
    }  
}
```

## 실습

- <국가, 수도>의 쌍 <한국, 서울>, <중국, 베이징>, <일본, 도쿄>들을 HashMap 객체로 생성 후 출력하시오. (국가명, 수도명을 각각 key, value로 설정)

## 실습

- 클래스 User는 email(메일주소, 문자열) 필드, name(이름, 문자열) 필드, name과 email을 파라미터로 전달 받는 생성자, name과 email 값을 반환하는 toString()으로 구성된다. 클래스 User를 작성하시오.
- yhlee@ee.ks.ac.kr(이영희), gdhong@ie.ks.ac.kr(홍길동), yhpark@cs.ks.ac.kr(박영희)에 해당하는 User 객체들을 HashMap 객체로 생성 후 출력하시오. (메일주소, User 객체를 각각 key, value로 설정)


```
public class Student {  
    String id;  
    int score;  
    public Student(String id, int score) {  
        this.id=id;  
        this.score=score;  
    }  
    public String getId() {  
        return id;  
    }  
    @Override  
    public String toString() {  
        return "<"+id+", "+score+">";  
    }  
}
```

```
public class Test {  
    public static void main(String[] args) {  
        Student s1=new Student("s-001", 98);  
        Student s2=new Student("s-002", 100);  
        HashMap<String, Student> map=new HashMap<>();  
        map.put(s1.getId(), s1);  
        map.put(s2.getId(), s2);  
        System.out.println(map);  
    }  
}
```

## References

 <http://docs.oracle.com/javase/7/docs/api/>

 <https://docs.oracle.com/javase/tutorial/java/>

 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.

 남궁성. 자바의 정석. 도우출판.

 황기태, 김효수 (2015). 명품 Java Programming. 생능출판사.