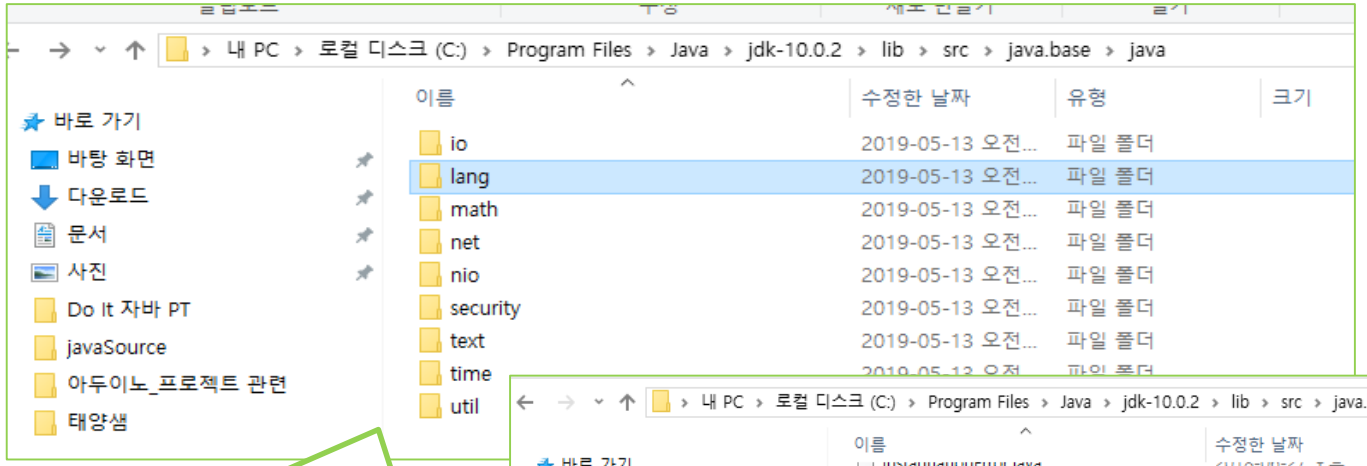
8장. 기본클래스, 예외처리, 추상클래스

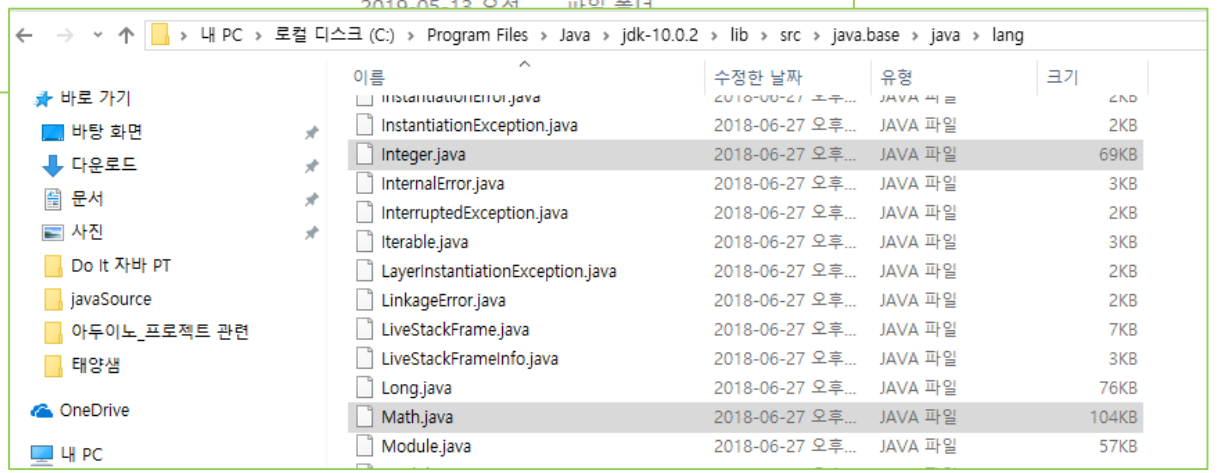
abstract class

java lang 패키지

- **java.lang 패키지**
  - Java.lang.Object / java.lang.String은 어디에 위치할까?
  - 포함된 클래스와 인터페이스는 import없이 사용할 수 있다.



자바 설치 경로에서 src.zip 압축풀기



java lang 패키지

- **주요 클래스**

클 래 스	용 도
Object	- 자바 클래스의 최상위 클래스로 사용
System	- 표준 입력 장치(키보드)로부터 데이터를 입력받을때 사용 - 표준 출력 장치(모니터)로 출력하기 위해 사용 - 자바 가상 기계를 종료시킬때 사용 - 쓰레기 수집기를 실행 요청할 때 사용
Class	- 클래스를 메모리로 로딩할 때 사용
String	- 문자열을 저장하고 여러 가지 정보를 얻을 때 사용
StringBuffer, StringBuilder	- 문자열을 저장하고 내부 문자열을 조작할 때 사용
Math	- 수학함수를 이용할 때 사용
Wrapper : Byte, Short, Character, Integer, Float, Double	- 기본 타입의 데이터를 갖는 객체를 만들 때 사용 - 문자열 기본타입으로 변환할때 사용 - 입력값 검사에 사용

Object 클래스

- **모든 클래스의 최상위 클래스 Object**
  - Java.lang.Object
  - 모든 클래스는 Object 클래스로부터 상속을 받는다. (컴파일러가 자동 변환)

```
class Book {
    int bookNumber;
    String bookTitle;
}
```

코드 작성할때

생략되어 있음

```
class Book extends Object {
    int bookNumber;
    String bookTitle;
}
```

컴파일러가 변환

Object 클래스

- **Object 클래스의 주요 메서드**

메서드	용 도
String **toString()**	객체를 문자열로 표현하여 반환한다. 재정의하여 객체에 대한 설명이나 특정 멤버 변수 값을 반환한다.
boolean **equals(Object obj)**	두 인스턴스가 동일한지 여부를 반환한다. 재정의하여 **논리적으로 동일한 인스턴스**임을 정의 할 수 있다.
int **hashCode()**	객체의 해시 코드 값을 반환한다.
Object **clone()**	객체를 복제하여 동일한 멤버 변수 값을 가진 새로운 인스턴스를 생성한다.
Class **getClass()**	객체의 클래스를 반환합니다.

Object 클래스

- **Object 클래스의 주요 메서드**

Module java.base
Package java.lang

Class Object

java.lang.Object

public class **Object**

Class Object is the root of the c

All Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
protected Object	clone()	Creates and returns a copy of thi
boolean	equals(Object obj)	Indicates whether some other ob
protected void	finalize()	**Deprecated.** The finalization mechanism is inh
Class<?>	getClass()	Returns the runtime class of this
int	hashCode()	Returns a hash code value for the
void	notify()	
void	notifyAll()	
String	toString()	

toString

```
public String toString()
```

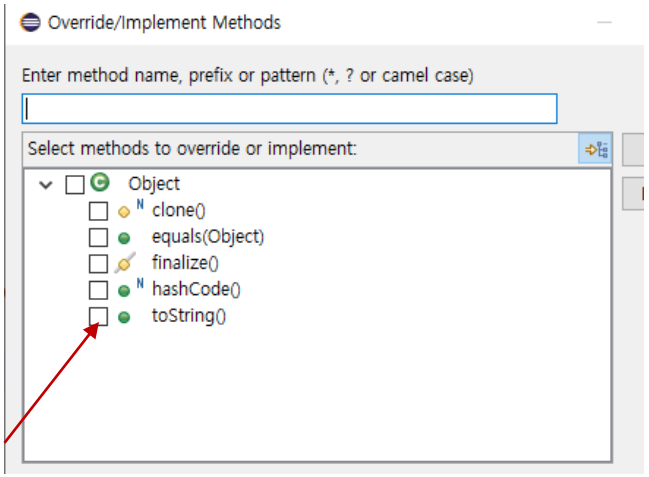
Returns a string representation of the object. In general, the toStr
that is easy for a person to read. It is recommended that all subclas

The toString method for class Object returns a string consisting
the hash code of the object. In other words, this method returns a s

toString() 메서드

- **toString() 메서드**
  - 객체 정보를 문자열로 바꾸어 줌
  - Integer나 String등 클래스는 toString() 메서드가 이미 재정의 되어 있음.
  - 사용자 정의 클래스는 toString() 재정의 해야함

```
package object;

public class Book {
    int bookNumber;
    String bookTitle;

    Book(int bookNumber, String bookTitle){
        this.bookNumber = bookNumber;
        this.bookTitle = bookTitle;
    }

    @Override
    public String toString() {
        return bookNumber + ", " + bookTitle;
    }
}
```



toString() 메서드

- **toString() 메서드**

```java
public class ToStringEx {

    public static void main(String[] args) {
        //String으로 생성한 인스턴스는 문자열로 출력
        String name = new String("홍길동");
        System.out.println(name);
        System.out.println(name.toString());

        //Book으로 생성한 문자열은 toString() 재정의 해야함
        Book book = new Book(100, "개미");
        System.out.println(book);
        System.out.println(book.toString());
    }
}
```

```java
public String toString() {
    return this;
}
```

이미 재정의 되어있음

```
홍길동
홍길동
개미,100
개미,100
```

equals() 메서드

- **equals() 메서드**
  - 두 인스턴스의 주소 값을 비교하여 boolean 값(true/false)를 반환
  - 재정의하여 두 인스턴스가 논리적으로 동일함의 여부를 반환.
  - 같은 번호의 책인 경우 여러 인스턴스의 주소값은 다르지만, 같은 책으로 인정할 수 있으므로 equals() 메서드를 재정의해야 함.

```
public class EqualsTest {
    public static void main(String[] args) {
        //String으로 생성한 인스턴스의 메모리 주소와 값 비교
        String name1 = new String("홍길동");
        String name2 = new String("홍길동");
        System.out.println(name1==name2);  //메모리 주소는 다르다.
        System.out.println(name1.equals(name2)); //저장된 값은 같다.

        //Book으로 생성한 인스턴스의 메모리 주소와 값 비교
        Book book1 = new Book(100, "개미");
        Book book2 = new Book(100, "개미");
        System.out.println(book1==book2);
        System.out.println(book1.equals(book2));
    }
}
```

```
false
true
=======
false
false
```

Book 클래스에서는 equals() 메서드 재정의 필요

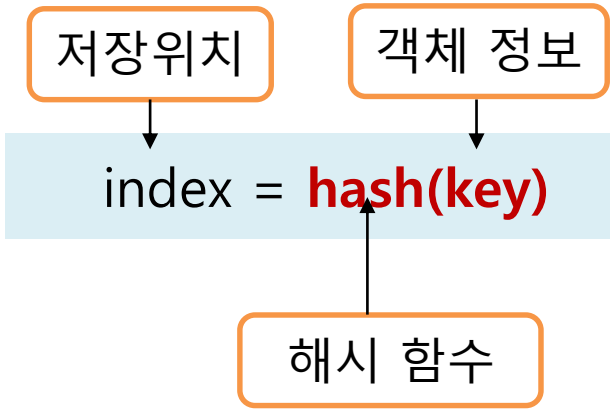
equals() 메서드

- equals() 재정의(Override)

```
public class Book {
    int bookNumber;
    String bookTitle;

    Book(int bookNumber, String bookTitle){
        this.bookNumber = bookNumber;
        this.bookTitle = bookTitle;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Book) {
            Book book = (Book)obj;  //강제 타입 변환
            if(this.bookNumber == book.bookNumber)
                return true;
        }
        return false;
    }
```

hashCode() 메서드

- **hashCode() 메서드**
  - hash : 정보를 저장, 검색하기 위해 사용하는 자료 구조
  - 자료의 특정 값(키 값)에 대해 저장 위치를 반환해 주는 해시 함수를 사용함. hash 알고리즘은 검색(search)에 효율적임
  - **hashCode() 메서드는 인스턴스의 저장 주소를 반환함 : 10진수로 나타냄**
  - 힙 메모리에 인스턴스가 저장되는 방식이 hash임

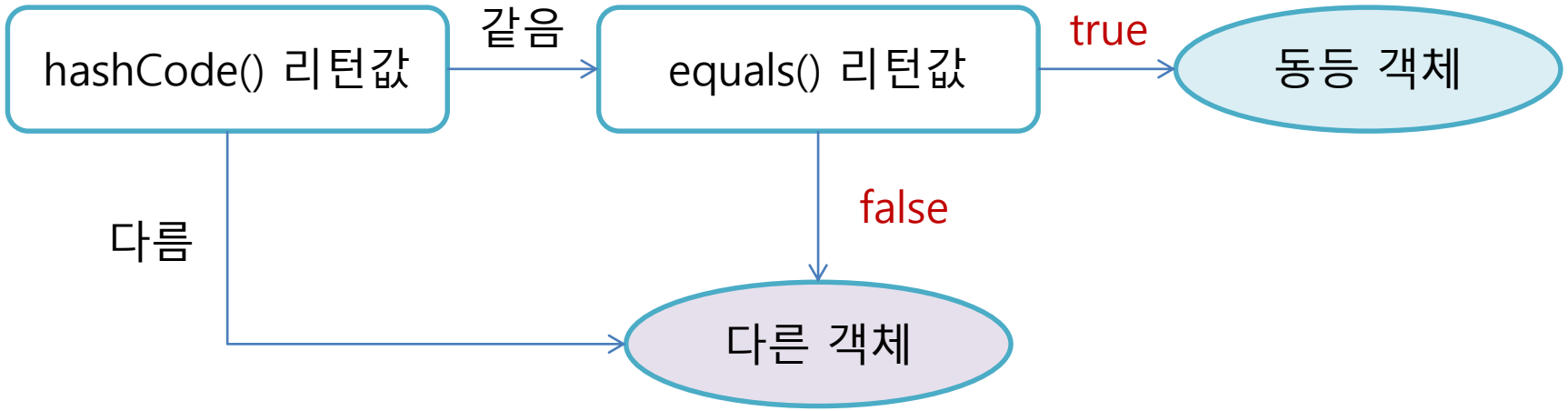


hashCode() 메서드

- **hashCode() 메서드**
  - hashCode()의 반환값: 자바 가상 머신이 저장한 인스턴스의 주소값을 10진수로 나타냄
  - 서로 다른 메모리의 두 인스턴스가 같다면 재정의 된 equals() 메서드의 값은 true이고, 동일한 hashCode() 값을 가져야 함
  - 논리적으로 동일함을 위해 equals() 메서드를 재정의했다면 hashCode(0 메서드도 재정의하여 동일한 값이 반환되도록 함

  - String 클래스: 동일한 문자열 인스턴스에 대해 동일한 정수가 반환됨
  - Integer 클래스: 동일한 정수값의 인스턴스에 대해 같은 정수값이 반환됨

객체 동등 비교



equals() 메서드

```java
package object.member;

public class Member {
    String id;

    Member(String id) {
        this.id = id;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj instanceof Member) {
            Member member = (Member)obj;
            if(id.equals(member.id)) {
                return true;
            }
        }
        return false;  //생성자 매개값 id와 일치하지 않으면
    }

    @Override
    public int hashCode() {
        return id.hashCode();
    }
}
```

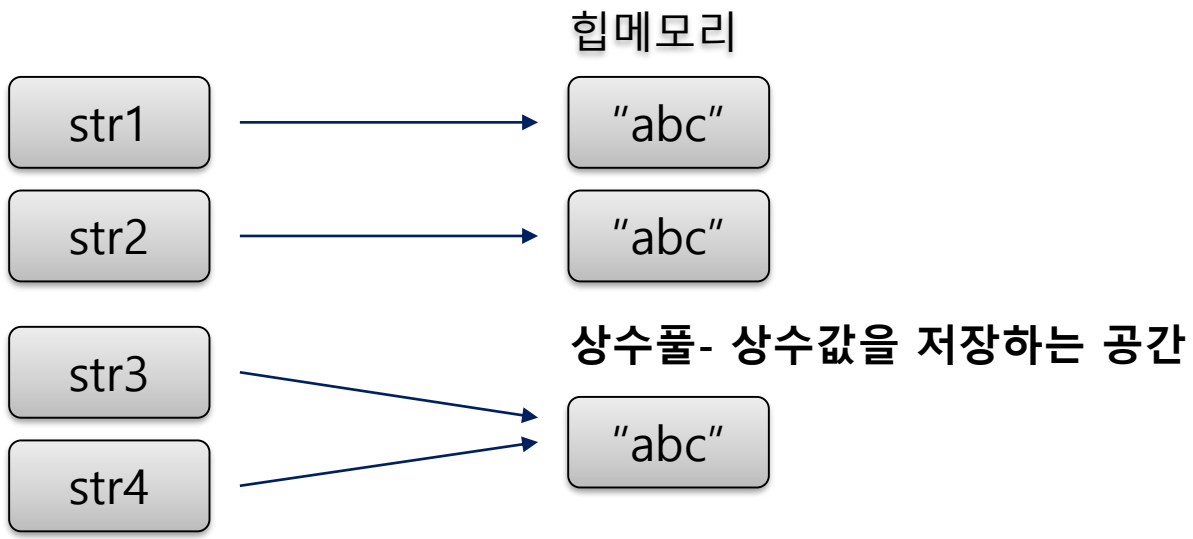
equals() 메서드

```
public class MemberTest {

    public static void main(String[] args) {

        Member obj1 = new Member("blue");
        Member obj2 = new Member("blue");
        Member obj3 = new Member("red");

        if(obj1.equals(obj2)) {
            System.out.println("obj1과 obj2는 일치합니다.");
        }else {
            System.out.println("obj1과 obj2는 다릅니다.");
        }

        if(obj1.equals(obj3)) {
            System.out.println("obj1과 obj은3 일치합니다.");
        }else {
            System.out.println("obj1과 obj3은 다릅니다.");
        }
    }
}
```

String 클래스

String 클래스

- 문자열을 저장하고 여러 가지 정보를 얻을 때 사용하는 클래스
- 문자열을 사용하여 생성자의 매개변수로 하여 생성하는 방식과 이미 생성된 문자열 상수를 가리키는 방식이 있다.



String 클래스

- **String 객체 생성**

```
public class StringTest {
    public static void main(String[] args) {

        String str1 = new String("abc");
        String str2 = new String("abc");

        //인스턴스가 매번 새로 생성되므로 주소값이 다름
        System.out.println(str1==str2);
        //문자열 값은 같음
        System.out.println(str1.equals(str2));

        String str3 = "abc";
        String str4 = "abc";

        //문자열 "abc"는 상수 풀에 저장되어 있어서 주소 값이 같음
        System.out.println(str3==str4);
        System.out.println(str3.equals(str4));
    }
}
```

```
false
true
true
true
```

String 클래스

- **charAt() – 문자추출**

  매개값으로 주어진 인덱스의 문자를 리턴함

```java
//charAt() 메서드 - 매개값의 문자 추출
String msg = "행운을 빌어요!!";

char ch = msg.charAt(0);
System.out.println(ch);

//주민등록번호에서 남여를 구분
String jumin = "020815-4234567";
char gender = jumin.charAt(7);  //성별

switch(gender) {
case '1': case 3:
    System.out.println("남자입니다.");
    break;
case '2': case '4':
    System.out.println("여자입니다.");
    break;
default:
    System.out.println("지원되지 않는 기능입니다.");
    break;
}
```

String 클래스

- **indexOf() – 문자열 찾기**

  매개값으로 주어진 문자열이 시작되는 인덱스를 리턴합니다.

```java
// 매개값이 없으면 -1을 리턴함
String subject = "자바 프로그래밍 입문";
int location1 = subject.indexOf("프로그래밍");
System.out.println(location1);

int location2 = subject.indexOf("코딩");
System.out.println(location2); //-1

//문자열 검색
if(subject.indexOf("자바") != -1) {
    System.out.println("자바와 관련된 책이군요!!");
}else {
    System.out.println("자바와 관련이 없는 책이군요!!");
}
```

String 클래스

- **substring() – 문자열 잘라내기**

  주어진 인덱스에서 문자열을 추출합니다.

```
//substring(beginIdx, endIdx)
//매개값의 시작 인덱스부터 끝(끝-1)까지 문자 추출
String juminNum = "991125-1234567";

String firstNum = juminNum.substring(0, 6);
System.out.println(firstNum);

String secondNum = juminNum.substring(7);
System.out.println(secondNum);
```

```
행
여자입니다.
3
-1
자바와 관련된 책이군요!!
991125
1234567
```

Wrapper 클래스

- **기본 자료형을 위한 클래스**
  - 기본 자료형을 멤버 변수로 포함하여 메서드를 제공함으로써 기본자료형의 객체를 제공하는 클래스.
  - 기본 자료형을 감쌌다는 의미로 Wrapper 클래스라고 한다.

지금까지 정수를 사용할 때 기본형인 int를 사용했다. 그런데 정수를 객체형으로 사용해야 하는 경우가 발생한다.

```
public void setValue(Integer i){.....}
//객체를 매개변수로 받는 경우

public Integer returnValue(){.....}
//반환값이 객체인 경우
```

기본형	Wrapper 클래스
boolean	Boolean
byte	Byte
char	Character
int	Integer
double	Double

Wrapper 클래스

- **자료형의 크기**

```
public class TypeSize {

    public static void main(String[] args) {
        //자료형의 크기 확인
        System.out.printf("byte형의 크기       ==> %d\n", Byte.SIZE);
        System.out.printf("short형의 크기      ==> %d\n", Short.SIZE);
        System.out.printf("int형의 크기        ==> %d\n", Integer.SIZE);
        System.out.printf("long형의 크기       ==> %d\n", Long.SIZE);
        System.out.printf("float형의 크기      ==> %d\n", Float.SIZE);
        System.out.printf("double형의 크기     ==> %d\n", Double.SIZE);
        System.out.printf("char형의 크기       ==> %d\n", Character.SIZE);
    }
}
```

```
byte형의 크기       ==> 8
short형의 크기      ==> 16
int형의 크기        ==> 32
long형의 크기       ==> 64
float형의 크기      ==> 32
double형의 크기     ==> 64
char형의 크기       ==> 16
```

Wrapper 클래스

- **Integer 클래스**

```java
public class IntegerTest {

    public static void main(String[] args) {
        //int형 < Integer형 (자동 형변환)
        Integer num1 = 100;
        int num2 = 200;
        int sum;

        //intValue() -> Integer형을 int형으로 변환함
        sum = num1.intValue() + num2;
        System.out.println(sum);

        //valueOf() -> 정수나 문자열을 Integer 클래스로 변환함
        Integer n1 = Integer.valueOf("100");
        System.out.println(n1);

        //parseInt() -> 매개로 전달된 문자열을 정수형으로 변환
        int n2 = Integer.parseInt("200");
        System.out.println(n2);

        //parseDouble() -> 문자열을 실수형으로 변환함
        double dNum = Double.parseDouble("1.609");
        System.out.println(dNum);
    }
}
```

```
300
100
200
1.609
```

에러와 예외

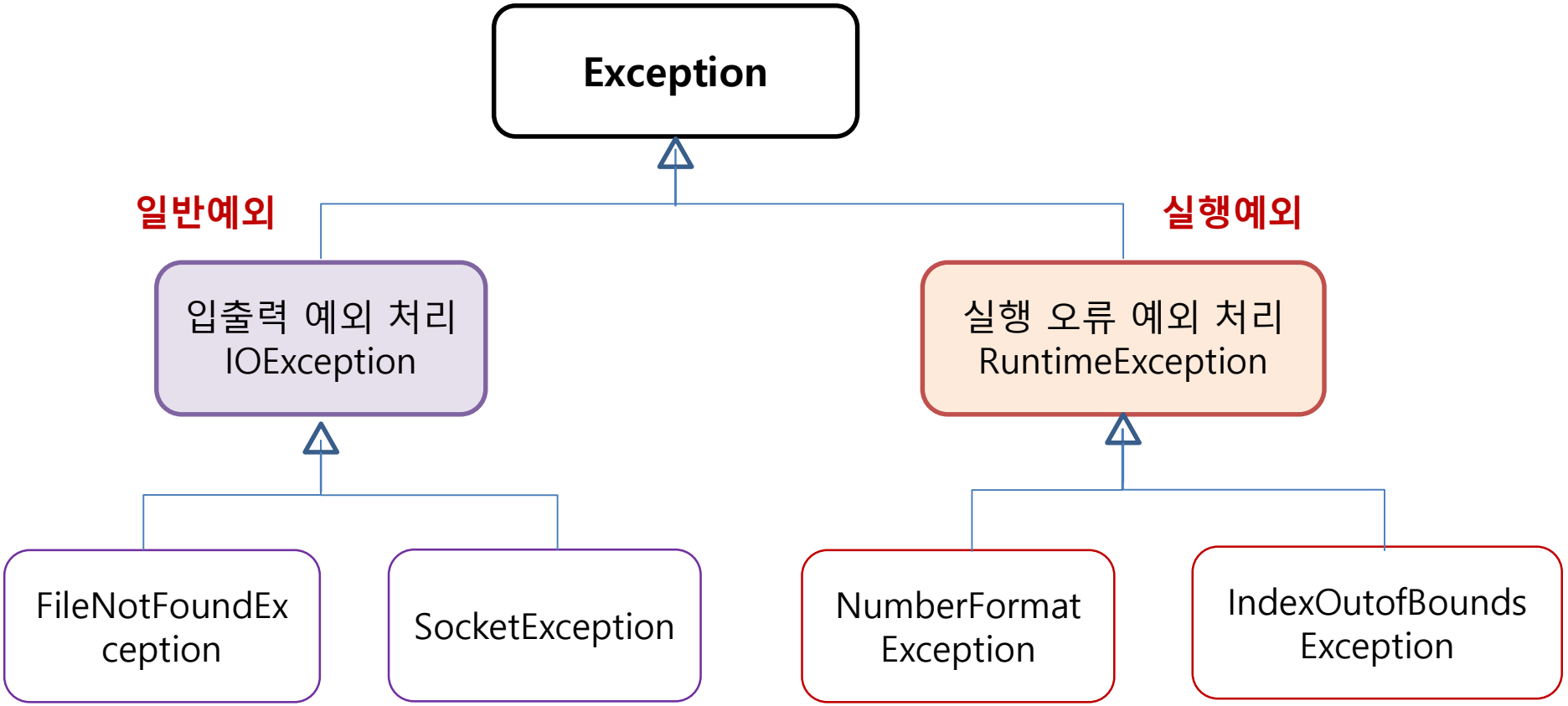
✓ 오류의 종류

1. **에러(Error)**
   - 하드웨어의 오작동 고장으로 인한 오류
   - 에러가 발생되면 프로그램이 종료되고 정상으로 돌아갈수 없음
2. **예외(Exception)**
   - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
   - 예외가 발생되면 프로그램이 종료되고, 예외 처리를 하면 정상으로 돌아갈 수 있음

✓ 예외의 종류

1. **일반 예외(Exception)**
   - 예외 처리가 없으면 컴파일 되지 않는 예외 – 컴파일 체크
2. **실행 예외(RuntimeException)**
   - 예외 처리를 생략해도 컴파일 되는 예외
   - 개발자의 경험과 판단으로 예외 코드 작성 필요

예외 클래스의 종류


Exception
일반예외
실행예외
입출력 예외 처리
IOException
실행 오류 예외 처리
RuntimeException
FileNotFoundException
SocketException
NumberFormatException
IndexOutofBoundsException

예외 클래스의 종류

- Java.lang 패키지 -> Exception Summary

Exception Summary

Exception	Description
ArithmeticException	Thrown when an exceptional arithmetic condi
ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been acc
ArrayStoreException	Thrown to indicate t
ClassCastException	Thrown to indicate t
ClassNotFoundException	Thrown when an app
CloneNotSupportedException	Thrown to indicate t
EnumConstantNotPresentException	Thrown when an app
Exception	The class Exception

Module java.base
Package java.lang

Class ArithmeticException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.ArithmeticException

All Implemented Interfaces:

Serializable

try ~ catch문

- **try ~ catch문**

  예외처리를 하면 예외 상황을 알려 주는 메시지를 볼 수 있고, 프로그램이 비정상적으로 종료되지 않고 계속 수행되도록 만들 수 있다.

```
try{
    예외가 발생할 수 있는 코드
}catch(처리할 예외 타입 e){
    예외를 처리하는 코드
}
```

실행 예외

- **실행 예외 – 컴파일러가 체크해주지 않음(NullPointerException)**

```
3  public class ExceptionHandling1 {
4
5      public static void printLength(String data) {
6          int result = data.length();
7          System.out.println("문자 수: " + result);
8      }
9
10     public static void main(String[] args) {
11         System.out.println("[프로그램 시작]\n");
12
13         printLength("Hello");
14         printLength(null);
15
16         System.out.println("[프로그램 종료]");
17     }
18 }
```

예외가 발생하면 그 즉시 프로그램이 종료됨

```
[프로그램 시작]

문자 수: 5
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String
        at exceptions.ExceptionHandling1.printLength(ExceptionHandling1.java:6)
        at exceptions.ExceptionHandling1.main(ExceptionHandling1.java:14)
```

실행 예외

- **실행 예외 처리 – try ~ catch 구문**

```java
public class ExceptionHandling2 {

    public static void printLength(String data) {
        //예외 처리: try ~ catch 구문
        try {
            int result = data.length();
            System.out.println("문자 수: " + result);
        }catch(NullPointerException e) {
            System.out.println(e.getMessage());
            e.printStackTrace();  //예외를 추적하면서 출력함(많이 사용됨)
        }
    }
```

예외가 발생하면 예외 처리 메시지가 출력되고 프로그램이 실행되고종료됨

```
[프로그램 시작]

문자 수: 5
Cannot invoke "String.length()" because "data" is null
java.lang.NullPointerException: Cannot invoke "String.length()" because "data"
        at exceptions.ExceptionHandling2.printLength(ExceptionHandling2.java:8)
        at exceptions.ExceptionHandling2.main(ExceptionHandling2.java:20)
[프로그램 종료]
```

일반 예외

- 일반 예외 – 컴파일러가 체크 함

```
public class ExceptionHandling3 {

    public static void main(String[] args) {

        Class.forName("java.lang.String");
    }
}
```

Unhandled exception type ClassNotFoundException

2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

Surround with try/catch 선택

일반 예외

- 일반 예외 – 컴파일러 체크

```
public class ExceptionHandling2 {

    public static void main(String[] args) {
        try {
            Class.forName("java.lang.String2");
            System.out.println("찾는 클래스가 있습니다.");
        } catch (ClassNotFoundException e) {
            System.out.println("클래스를 찾을 수 없습니다.");
            e.printStackTrace();
        }
    }
}
```

```
클래스를 찾을 수 없습니다.
java.lang.ClassNotFoundException: java.lang.String2
        at java.base/jdk.internal.loader.BuiltinClassLoader.l
        at java.base/jdk.internal.loader.ClassLoaders$AppClas
        at java.base/java.lang.ClassLoader.loadClass(ClassLoa
        at java.base/java.lang.Class.forName0(Native Method)
```

try~catch~finally문

- **try~catch~finally문 사용하기**

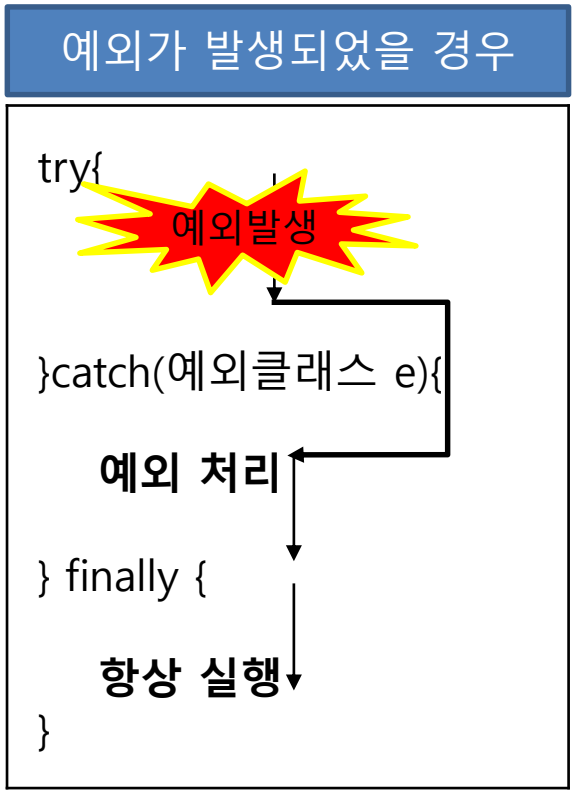
  프로그램에서 외부장치와의 연동시 초기화나 마무리 작업시 주로 사용한다.

  이때 사용하는 블록이 finally인데 일단 try블록이 수행되면 어떤 경우에도 반드시 수행된다.

정상실행 되었을 경우

```
try{
    예외 발생 가능 코드
}catch(예외클래스 e){
    예외 처리
} finally {
    항상 실행
}
```

예외가 발생되었을 경우



try~catch~finally문

- **try~catch~finally문 사용하기**

```
        System.out.println("============================");

        try {
            Class.forName("java.lang.String2");
            System.out.println("찾는 클래스가 있습니다.");
        } catch (ClassNotFoundException e) {
            System.out.println("클래스를 찾을 수 없습니다.");
            e.printStackTrace();
        } finally {
            System.out.println("항상 수행됨");
        }
}
```

```
클래스를 찾을 수 없습니다.
java.lang.ClassNotFoundException: java.lang.String2
        at java.base/jdk.internal.loader.BuiltinClass
        at java.base/jdk.internal.loader.ClassLoaders
        at java.base/java.lang.ClassLoader.loadClass(
        at java.base/java.lang.Class.forName0(Native
        at java.base/java.lang.Class.forName(Class.ja
        at java.base/java.lang.Class.forName(Class.ja
        at Chapter8/exceptions.ExceptionHandling3.mai
항상 수행됨
```

다중 예외 처리

- 다중 예외 – 동시에 여러 개의 예외가 발생한 경우

```
public class MultiException {

    public static void main(String[] args) {

        String[] array = {"100", "123a"};

        for(int i = 0; i <= array.length; i++) {

            System.out.println(array[i]); //배열 출력
            //문자열을 정수로 변환
            int value = Integer.parseInt(array[i]);
            System.out.println(value);

        }
    }
}
```

다중 예외 처리

- **다중 예외 처리**

[java.lang.ArrayIndexOutOfBoundsException]

```
100
123a
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
        at Chapter8/exceptions.MultiException.main(MultiException.java:11)
```

[java.lang.NumberFormatException]

```
100
100
123a
Exception in thread "main" java.lang.NumberFormatException: For input string: "123a"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:662)
        at java.base/java.lang.Integer.parseInt(Integer.java:778)
        at Chapter8/exceptions.MultiException.main(MultiException.java:13)
```

다중 예외 처리

- 다중 예외 처리

```
public class MultiException {

    public static void main(String[] args) {

        String[] array = {"100", "123a"};

        for(int i = 0; i <= array.length; i++) {
            try {
                System.out.println(array[i]); //배열 출력
                //문자열을 정수로 변환
                int value = Integer.parseInt(array[i]);
                System.out.println(value);
            }catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("배열 인덱스가 초과되었습니다.");
            }catch(NumberFormatException e) {
                System.out.println("숫자로 변환할 수 없는 항목이 있습니다.");
            }
        }
    }
}
```

```
100
100
123a
숫자로 변환할 수 없는 항목이 있습니다.
배열 인덱스가 초과되었습니다.
```

예외 처리 미루기 – throws

- **throws 로 예외처리 미루기(떠넘기기)**

  예외 처리를 해당 메서드에서 하지 않고 미룬 후, 메서드를 호출하여 사용하는 곳에서 예외를 처리하는 방법이다.

```
메서드명 throws 예외클래스1, 예외클래스2,..{
}
```

```
public static void findClass() {
    Class.forName("java.lang.Math");
}
```

Unhandled exception type ClassNotFoundException

2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

Add throws declaration 선택

예외 처리 – throws

- **throws 로 예외처리 미루기(떠넘기기)**

```
public class ThrowsExample {

    public static void main(String[] args) {
        //호출(사용)한 곳에서 예외 처리함
        try {
            findClass();
        } catch (ClassNotFoundException e) {
            //e.printStackTrace();
            System.out.println("예외 처리: " + e.toString());
        }
    }

    public static void findClass() throws ClassNotFoundException {
        //예외를 미룸
        Class.forName("java.lang.Math2");
    }
}
```

```
예외 처리: java.lang.ClassNotFoundException: java.lang.Math2
```

추상 클래스(abstract class)

❖ 추상 클래스

객체를 직접 생성할 수 있는 클래스를 실체 클래스라고 한다면 이 클래스들의 공통적인 특성을 추출해서 선언한 클래스를 추상 클래스라 한다.

추상클래스와 실체 클래스는 상속 관계를 구성한다.
왜 추상클래스를 사용하는가?
실체 클래스의 필드와 메서드의 이름을 통일할 목적으로 사용한다.

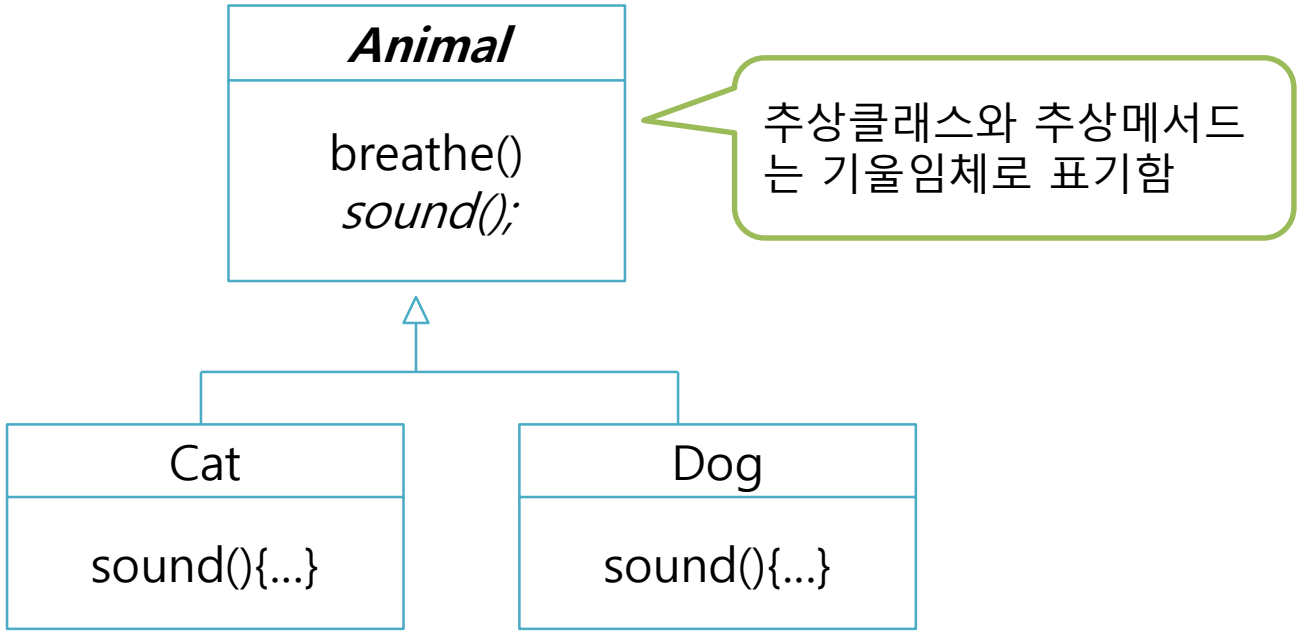
TelePhone 클래스 – owner(소유자), powerOn() - 전원을 켜다
SmartPhone 클래스 – user(소유자), trunOn() – 전원을 켜다

▪ 추상 클래스의 선언

```
public abstract class 클래스이름{
    //필드, 생성자, 메서드
}
```

추상 클래스(abstract class)

- **추상 메서드**
  - 추상 메서드도 abstract 예약어를 사용한다.
  - 메서드를 구현하지 않고 선언만 한다. { } 구현부가 없다.
  - 상속받는 실체 클래스는 추상메서드를 필수적으로 구현해야 한다.



추상 메서드

- **추상 메서드 선언**

```
package abstract_class.animal;

public abstract class Animal {

    String kind; //동물의 종

    void breathe() {
        System.out.println("동물이 숨을 쉽니다.");
    }

    //추상 메서드 선언
    public abstract void cry();

}
```

추상 메서드

- 동물의 소리를 구현한 추상클래스 상속 예.

```
package abstract_class.animal;

public class Cat extends Animal{

    public Ca
        this.
    }


}
```

The type Cat must implement the inherited abstract method Animal.cry()
2 quick fixes available:
Add unimplemented methods
Make type 'Cat' abstract
Press 'F2' for focus

추상메서드는 반드시 구현해야 함

```
public class Cat extends Animal{

    public Cat() {
        this.kind = "포유류";
    }

    @Override
    public void cry() {
        System.out.println("야~ 옹!");
    }
}
```

추상 메서드

- 동물의 소리를 구현한 추상클래스 상속 예.

```
public class Dog extends Animal{

    public Dog() {
        this.kind = "포유류";
    }

    @Override
    public void cry() {
        System.out.println("멍멍!");
    }
}
```

추상 메서드

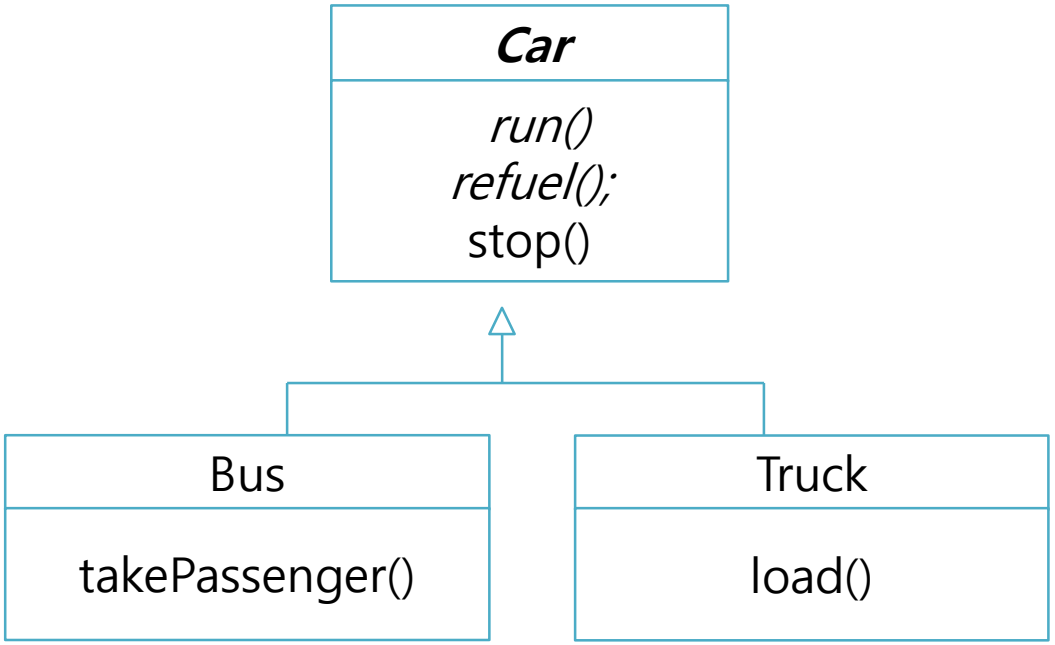
- 동물의 소리를 구현한 추상클래스 상속 예.

```
public class AnimalTest {

    public static void main(String[] args) {
        Cat cat = new Cat();
        cat.breathe();
        cat.cry();

        Dog dog = new Dog();
        dog.breathe();
        dog.cry();

        //메서드의 다형성
        animalCry(new Cat());
        animalCry(new Dog());

    }

    //동물의 울음소리 메서드 정의
    public static void animalCry(Animal animal) {
        animal.cry();
    }
}
```

```
동물이 숨을 쉽니다.
야~ 옹!
동물이 숨을 쉽니다.
멍멍!
야~ 옹!
멍멍!
```

추상 클래스(abstract class)

- **자동차를 구현한 추상 클래스 상속 예.**



추상클래스 실습

- 자동차를 구현한 추상 클래스 상속 예

```
public abstract class Car {
    public abstract void run();

    public abstract void refuel();

    public void stop() {
        System.out.println("차가 멈춥니다.");
    }
}
```

추상클래스 실습

- 자동차를 구현한 추상 클래스 상속 예

```
public class Bus extends Car{

    public void takePassenger() {
        System.out.println("승객을 버스에 태웁니다.");
    }

    @Override
    public void run() {
        System.out.println("버스가 달립니다.");
    }

    @Override
    public void refuel() {
        System.out.println("천연 가스를 충전합니다.");
    }
}
```

추상클래스 실습

- 자동차를 구현한 추상 클래스 상속 예

```
public class Truck extends Car{

    @Override
    public void run() {
        System.out.println("트럭이 달립니다.");
    }

    @Override
    public void refuel() {
        System.out.println("휘발유를 주유합니다.");
    }

    public void load() {
        System.out.println("짐을 싣습니다.");
    }
}
```

추상클래스 실습

- 자동차를 구현한 추상 클래스 상속 예

```
//Bus 객체 생성
Bus bus = new Bus();

bus.run();
bus.refuel();
bus.takePassenger();

//Truck 객체 생성
Truck truck = new Truck();

truck.run();
truck.refuel();
truck.load();
```

```
버스가 달립니다.
천연 가스를 충전합니다.
버스에 승객을 태웁니다.
트럭이 달립니다.
휘발유를 주유합니다.
트럭에 짐을 싣습니다.
```

final 예약어

- **상수를 의미하는 final 변수**
  - 해당 선언이 최종 상태이고, 결코 수정될 수 없음을 뜻한다.

```
package finalex;

public class Constant {
    int num = 10;
    final int NUM = 100;

    public static void main(String[] args) {
        Constant cons = new Constant();
        cons.num = 20;
        //cons.NUM = 1000; //final 상수이므로 변경할 수 없음

        System.out.println(cons.num);
        System.out.println(cons.NUM);
    }
}
```

final 상수

- 여러 파일에서 공유하는 상수

```
public class Define {
    public static final int MIN = 1;
    public static final int MAX = 99999;
    public static final int ENG = 1001;
    public static final int MATH = 2001;
    public static final double PI = 3.14;
    public static final String GOOD_MORNING = "Good Morning!";
}
```

final 상수

- 여러 파일에서 공유하는 상수

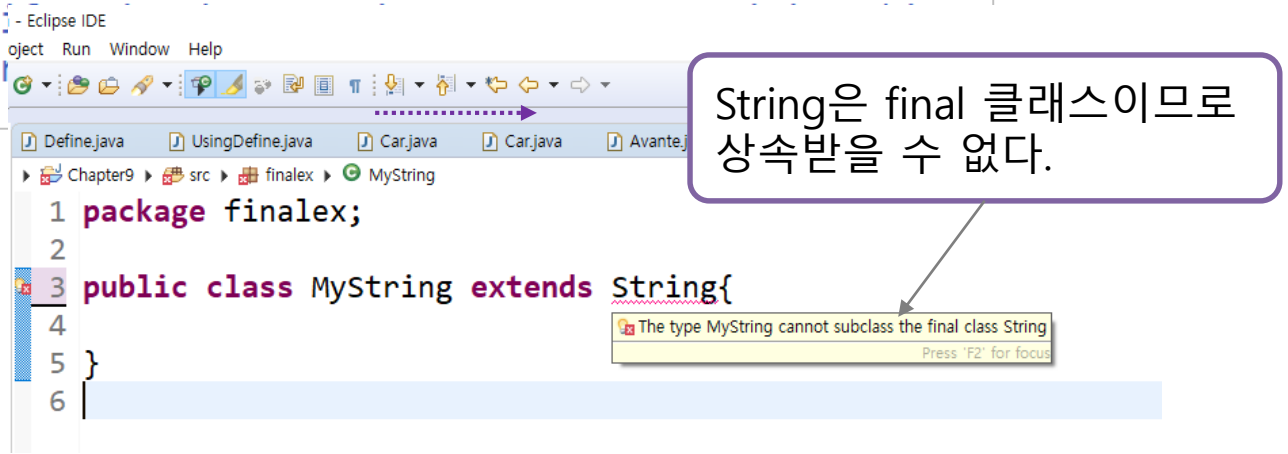
```
public class UsingDefine {

    public static void main(String[] args) {
        System.out.println(Define.GOOD_MORNING);
        System.out.println("최솟값은 " + Define.MIN + "입니다.");
        System.out.println("최대값은 " + Define.MAX + "입니다.");
        System.out.println("수학 과목 코드값은 " + Define.MATH + "입니다.");
        System.out.println("영어 과목 코드값은 " + Define.ENG + "입니다.");
    }
}
```

```
Good Morning!
최솟값은 1입니다.
최대값은 99999입니다.
수학 과목 코드값은 2001입니다.
영어 과목 코드값은 1001입니다.
```

final 클래스

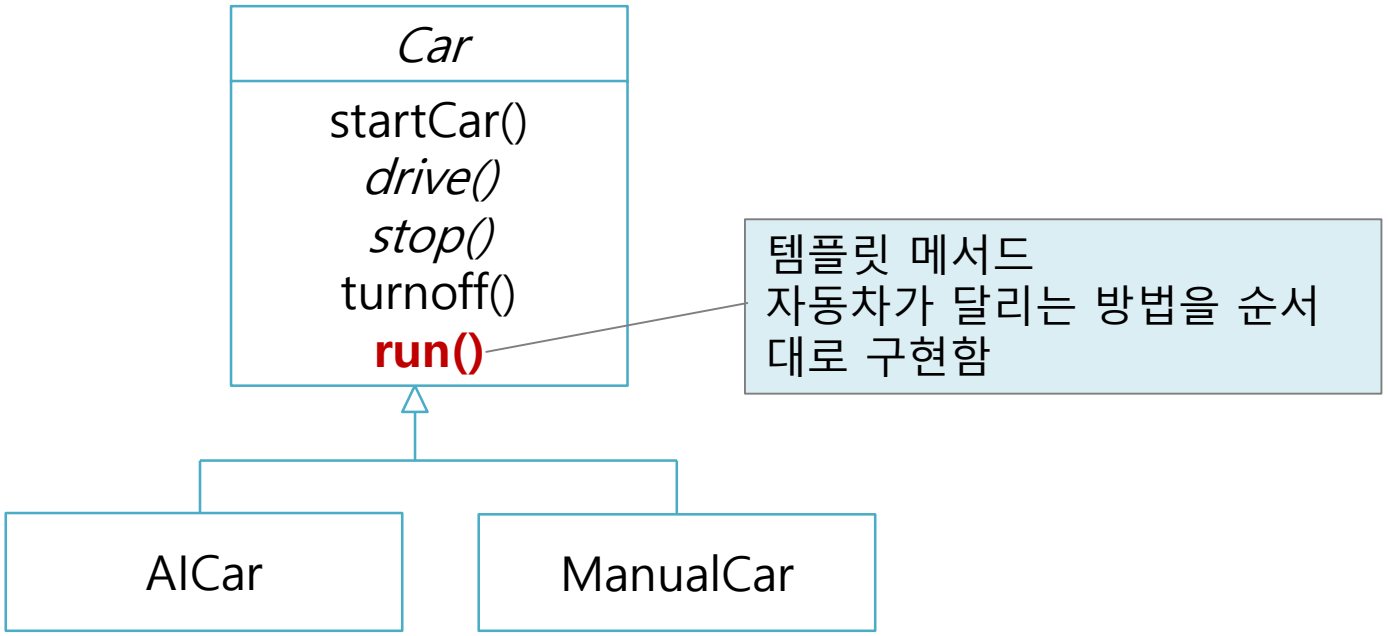
- **보안과 관련되어 있거나 기반클래스가 변하면 안 되는 경우**
  - ✓ String이나 Integer 클래스 등.

```
public final class String
    implements java.io.Serializable, Comparable<String>, CharSequence {

    /**
     * The value is used for character storage.
     *
     * @implNote This field is trusted by the VM, and is a subject to
     * constant folding
     * field after const
     *
```



템플릿 메서드

- **템플릿 메서드란?**
  - 템플릿 메서드 : 추상 메서드나 구현된 메서드를 활용하여 전체 기능의 흐름(시나리오)를 정의하는 메서드.
  - final로 선언하면 하위 클래스에서 재정의 할 수 없음



템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public abstract class Car {
    public abstract void drive();
    public abstract void stop();

    public void startCar() {
        System.out.println("시동을 켭니다.");
    }

    public void turnOff() {
        System.out.println("시동을 끕니다.");
    }

    public final void run() {
        startCar();
        drive();
        stop();
        turnOff();
    }
}
```

final로 선언
상속받은 하위 클래스가 메서드를 재정의 할 수 없다.

템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class HumanCar extends Car{

    @Override
    public void drive() {
        System.out.println("사람이 차를 운전합니다.");
    }

    @Override
    public void stop() {
        System.out.println("사람이 브레이크를 밟아 정지합니다.");
    }
}
```

템플릿 메서드

- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class AICar extends Car{

    @Override
    public void drive() {
        System.out.println("자동차가 자율 주행합니다.");
    }

    @Override
    public void stop() {
        System.out.println("자동차가 스스로 멈춥니다.");
    }
}
```

템플릿 메서드

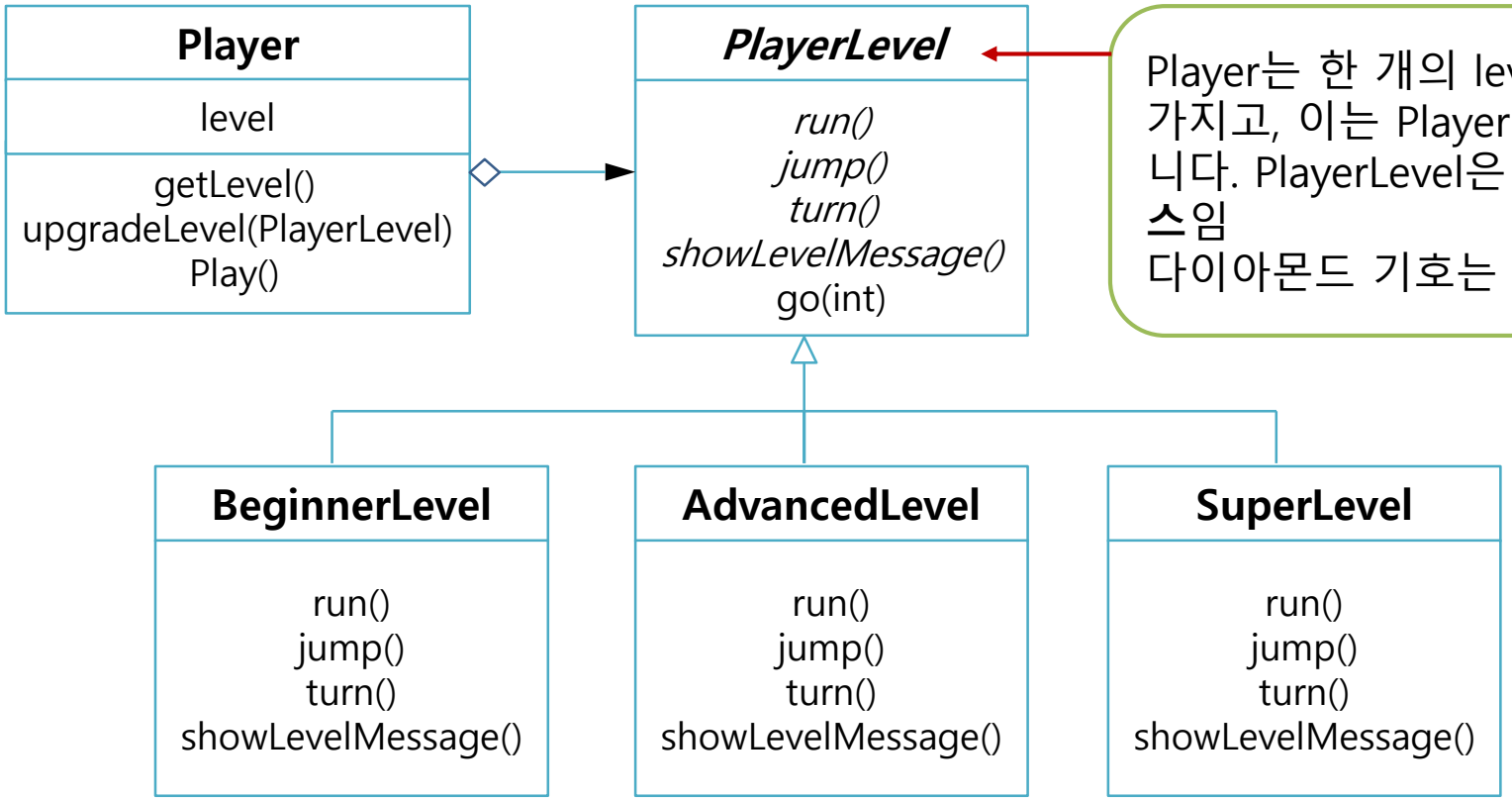
- 템플릿 메서드를 사용한 추상클래스 상속 예.

```
public class CarTest {

    public static void main(String[] args) {

        System.out.println("===== 사람이 운전하는 자동차 =====");
        Car hisCar = new HumanCar();
        hisCar.run();

        System.out.println("===== 자율 주행하는 자동차 =====");
        Car myCar = new AICar();
        myCar.run();
    }
}
```

```
===== 사람이 운전하는 자동차 =====
시동을 켭니다.
사람이 차를 운전합니다.
사람이 브레이크를 밟아 정지합니다.
시동을 끕니다.
===== 자율 주행하는 자동차 =====
시동을 켭니다.
자동차가 자율 주행합니다.
자동차가 스스로 멈춥니다.
시동을 끕니다.
```

Game Level App

예제 시나리오
Player가 있고 이 플레이어가 게임을 합니다. 게임에서 Player가 가지는 레벨에 따라 세가지 기능이 있는데 run(), jump(), turn()입니다.



Game Level App

PlayerLevel 클래스

각 레벨에서 수행할 공통 기능은 PlayLevel 추상 클래스에서 선언한다.

```
public abstract class PlayerLevel {
    public abstract void run();
    public abstract void jump();
    public abstract void turn();
    public abstract void showLevelMessage();

    final public void go(int count) {
        run();
        for(int i=0; i<count; i++) {
            jump();
        }
        turn();
    }
}
```

한번 run하고, count만큼 jump하고, 한번 turn한다.

Game Level App

Beginner 클래스

초보자 레벨에서는 천천히 달리 수만 있다. 점프나 턴을 할 수 없다.

```
public class Beginner extends PlayerLevel {

    @Override
    public void run() {
        System.out.println("천천히 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("jump할 줄 모르지롱.");
    }

    @Override
    public void turn() {
        System.out.println("Turn할 줄 모르지롱.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****초보자 레벨입니다.*****");
    }
}
```

Game Level App

Advanced 클래스

중급자 레벨에서는 빠르게 달릴 수 있고, 높이 점프할 수 있다. 턴을 할 수 없다.

```
public class AdvancedLevel extends PlayerLevel{

    @Override
    public void run() {
        System.out.println("빨리 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("높이 jump합니다.");
    }

    @Override
    public void turn() {
        System.out.println("Turn 할 줄 모르지롱.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****중급자 레벨입니다.*****");
    }
}
```

Game Level App

SuperLevel 클래스

고급자 레벨에서는 매우 빠르게 달릴 수 있고, 매우 높이 점프할 수 있다. 턴하는 기술도 사용할 수 있다.

```
public class SuperLevel extends PlayerLevel{

    @Override
    public void run() {
        System.out.println("매우 빨리 달립니다.");
    }

    @Override
    public void jump() {
        System.out.println("매우 높이 jump합니다.");
    }

    @Override
    public void turn() {
        System.out.println("한 바퀴 돕니다.");
    }

    @Override
    public void showLevelMessage() {
        System.out.println("*****고급자 레벨입니다.*****");
    }
}
```

Game Level App

Player 클래스

```
public class Player {
    //PlayerLevel 클래스 참조
    private PlayerLevel level;

    public Player() {
        level = new Beginner();
        level.showLevelMessage();
    }

    public void upgradeLevel(PlayerLevel level) {//매개변수의 다형성
        this.level = level;
        level.showLevelMessage();
    }

    public void play(int count) { //템플릿 메서드 호출
        level.go(count);
    }
}
```

부모 타입으로 객체 생성(다형성)

Game Level App

테스트 프로그램 실행

```
Player player = new Player();
//처음 생성시 BeginnerLevel
player.play(1);

//중급자 레벨
AdvancedLevel aLevel = new AdvancedLevel();
player.upgradeLevel(aLevel);
player.play(2);

//고급자 레벨
SuperLevel sLevel = new SuperLevel();
player.upgradeLevel(sLevel);
player.play(3);
```

```
*****초보자 레벨입니다.*****
천천히 달립니다.
jump할 줄 모르지롱.
Turn할 줄 모르지롱.
*****중급자 레벨입니다.*****
빨리 달립니다.
높이 jump합니다.
높이 jump합니다.
Turn 할 줄 모르지롱.
*****고급자 레벨입니다.*****
매우 빨리 달립니다.
매우 높이 jump합니다.
매우 높이 jump합니다.
매우 높이 jump합니다.
한 바퀴 돕니다.
```

실습 문제 1 – Object 클래스

다음 코드의 출력 결과가 "진돗개 백구"가 되도록 MyDog 클래스를 수정하세요.

```
class MyDog{
    String name;
    String type;

    public MyDog(String name, String type) {
        this.name = name;
        this.type = type;
    }

    ...

}

public class ToStringTest {

    public static void main(String[] args) {

        MyDog dog = new MyDog("백구", "진돗개");
        System.out.println(dog);
    }

}
```