edureka!

Method In Java

메소드(Method)에 대한 모든 것

KOREATECH 설순욱

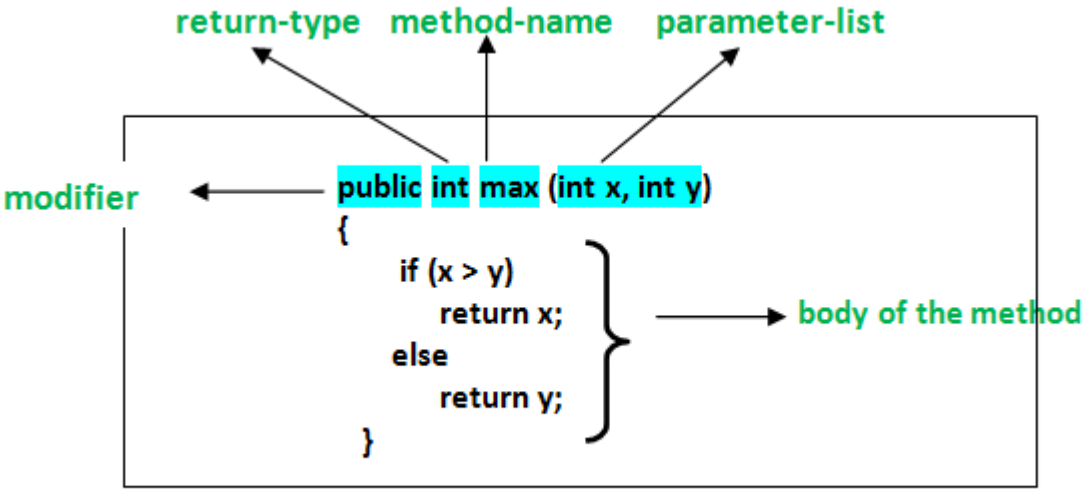
1

메소드의 기본

1. 메소드 정의 방법과 위치
2. 메소드의 호출
3. 매개변수 전달
4. 리턴 값

메소드(Method) 정의 방법과 위치

- 메소드 정의 방법
  - *수식어 리턴타입 메소드이름(매개변수) { 본문 코드 }*



  - 클래스 내에 포함되어야 함

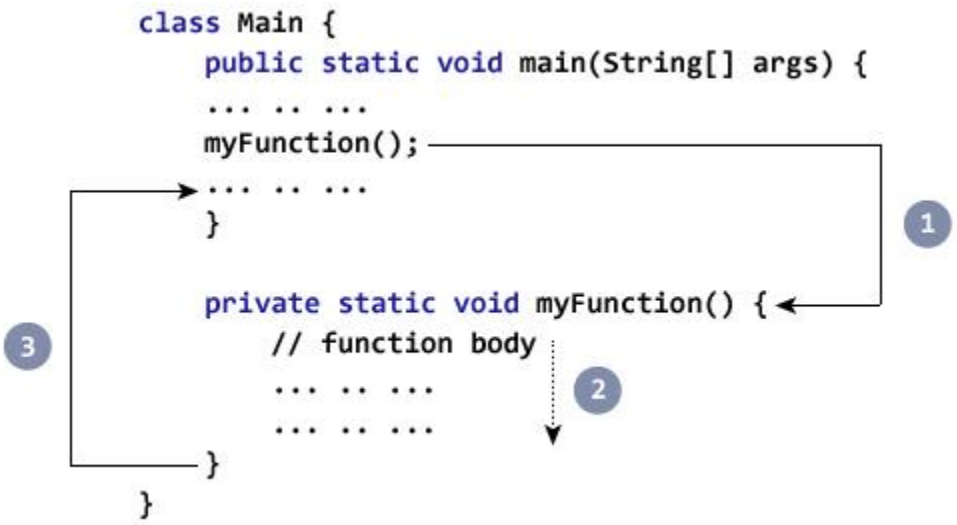
```
class MyClass{
    public int max(int x, int y) {
        return (x>y) ? x : y;
    }
}
```

메소드(Method) 정의 방법과 위치

- FAQ
  - 메소드 이름을 정하는 규칙이 있나요?
    - 개발자들의 원칙 있음. 동사, 소문자로 시작, 대문자연결 (run, runFast, getFinalData, setNumber, isEmpty)
  - 클래스 내에서 위치(순서)는 어디로 해야 하나요?
    - 클래스 내의 변수, 메소드, 생성자 들의 위치(순서)는 어디든 OK
    - 메소드 내에 다시 메소드를 정의할 수 없음
    - 클래스 밖에 존재할 수 없음
  - 메소드 이름 앞에 public, static은 무엇인가요?
    - 접근지정자로 public, private, protected, 혹은 아무것도 없이 설정하여 다른 패키지, 클래스, 자식클래스에서 호출 가능 여부를 결정
    - static으로 지정하면 Math.abs()처럼 객체를 만들지 않고 호출 가능
  - 같은 이름의 메소드를 여러 개 정의할 수 있나요?
    - 파라미터의 개수나 타입을 다르게 하여 가능➔오버로딩(overloading)
  - 부모 클래스에 이미 있는 메소드를 만들 수 있나요?
    - 상속받는 super 클래스에 있는 메소드를 재정의(overriding) 가능

메소드 호출: 실행 순서

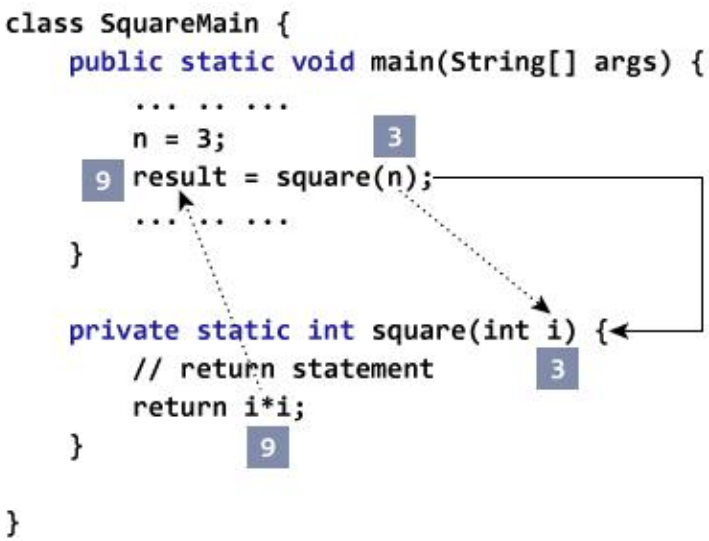
- "methodName()" 으로 호출하고 실행이 끝나면 돌아온다.



- 모든 프로그램의 시작 위치는 main() 메소드
  - main()에서 다른 메소드를 호출하고, 거기서 다시 다른 메소드 호출...
  - 모든 메소드 실행이 끝나고 main()도 끝나면 프로그램 종료
  - 단, 쓰레드(병렬/동시 처리)를 만들면 쓰레드의 run() 메소드가 종료되어야 함

매개변수와 리턴값

- 메소드에 값을 전달하고 결과 값을 받는 구조



- 매개변수(arguments): 파라미터(parameters), 인자 등으로 불림
  - 0개~n개 (any data type)
- 리턴 값
  - 0개: 함수에서 일을 처리하고 그냥 돌아옴
  - 1개: 처리 결과를 하나의 데이터 값으로 치환하는 효과
    - 예제에서, square(n)은 숫자 9로 치환되는 효과 ➔ result = 9;

매개변수 – 없을 때

- 매개변수가 없을 때
  - 괄호 안에 아무것도 쓰지 않으면 됨

```
public class Test {
    public static void main(String[] args) {
        sayHello();
    }
    private static void sayHello() {
        System.out.println("안녕하세요?!");
    }
}
```

매개변수 – 있을 때

- 어떤 타입의 값을 전달할 수 있나요?
  - 기본타입(int, double...), 배열, 객체(레퍼런스 타입)
  - 값(리터럴)을 전달하거나 변수 값을 전달

- 메소드 선언과 호출 방식
  - 괄호 안에 변수 선언하듯이 데이터타입과 변수 이름을 지정
    - 예, void sayHello(String name) { … }
  - 호출하는 쪽은 해당하는 타입의 변수나 값을 전달

```
public class Test {
    public static void main(String[] args) {
        String n = "이순신";
        sayHello(n);
        sayHello("홍길동");
    }
    private static void sayHello(String name) {
        System.out.println(name+"님 안녕하세요?!");
    }
}
```

매개변수 – 여러 개 전달

- 매개변수가 여러 개 있을 때
  - 쉼표(,)를 이용해서 여러 개의 변수를 선언
    - 예, void printNameRepeat(String name, int n) { … }
  - 호출하는 쪽은 쉼표(,)를 이용해서 여러 개의 값이나 변수를 전달

```
public class Test {
    public static void main(String[] args) {
        printNameRepeat("홍길동", 3);
    }
    private static void printNameRepeat(String name, int n) {
        for (int i = 0; i < n; i++) {
            System.out.println(name);
        }
    }
}
```

리턴값 – 없는 경우 void로 선언

- 메소드는 0~1개의 값을 리턴 가능
- 리턴할 내용이 없는 경우 void 타입으로 메소드를 정의해야 함

```
public class Test {
    public static void main(String[] args) {
        String n = "이순신";
        sayHello(n);
        sayHello("홍길동");
    }
    private static void sayHello(String name) {
        System.out.println(name+"님 안녕하세요?!");
    }
}
```

  - 출력 문장(println)이 있다고 해서 리턴 값이 있는 것은 아님

리턴값 – 없는 경우에도 return 사용 가능

- 리턴 타입이 void인 경우도 return 문장을 실행할 수 있나요?
  - YES
  - 특정 조건에서 메소드를 종료하고자 하는 경우 값 없이 return 문장만 사용

```
void printChar(char ch, int num) {
     if( num <=0 ) return;
     System.out.println(ch);
}
```

리턴값 – 기본 타입

- 리턴 값의 종류: 기본 타입, 배열, 객체
  - 기본 타입(primitive type)은 int, double, char 등을 데이터 타입
  - 배열은 같은 타입 여러 값을 전달하는 효과
  - 객체는 서로 다른 타입의 여러 값을 전달하는 효과
- 기본 타입의 값을 리턴하는 예제
  - return a+b; 처럼 리턴 문장 필수
  - sum() 메소드의 리턴 값을 sum 변수에 저장하고 있는데, 변수 이름과 메소드 이름은 같아도 충돌 없음. 괄호()가 있으면 메소드임

```
public class Test {
    public static void main(String[] args) {
        int sum = sum(10, 20);
    }
    private static int sum(int a, int b) {
        return a+b;
    }
}
```

리턴값 – 배열 타입

- 같은 타입의 값을 여러 개 전달하는 효과를 얻을 수 있음
- 실제는 배열 참조 값 1개를 리턴하는 것임
  - 아래 예제에서 makeArray() 메소드에서 배열을 생성(new int[size])하고 메모리 공간을 찾아갈 수 있는 참조 변수의 값을 리턴하는 것임

```
public class Test {
    public static void main(String[] args) {
        int[] arr = makeArray(5);
    }
    private static int[] makeArray(int size) {
        int[] arr = new int[size];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = (int)(Math.random() * 100);
        }
        return arr;
    }
}
```

리턴값 – 객체 타입1

- 여러 타입의 데이터 덩어리를 전달하는 효과
- 배열과 마찬가지로 참조 값 1개를 전달하는 것임
  - 아래 function() 메소드에서 Data 타입의 객체를 리턴함 (레퍼런스 값 리턴)

```
class Data{
    int x;
    String contnet;
}
public class Test {
    public static void main(String[] args) {
        Data d = function();
    }
    private static Data function() {
        Data data = new Data();
        data.x = 10;
        data.contnet = "내용";
        return data;
    }
}
```

리턴값 – 객체 타입2

- 복잡한 데이터 타입도 만들어 사용 가능
  - 아래 예제에서 createData() 메소드의 리턴 값은 DataRecord 타입

```
class Address{ /*코드 생략 */}
class DataRecord {
    int id;
    double scores[];
    String type, courses[];
    Person father, mother, friends[];
    Address home, work;
}
public class MethodEx {
    public static DataRecord createData() {
        DataRecord record = new DataRecord();
        // 데이터 입력받기..
        return record;
    }
```

리턴값 – 객체 배열 타입

- 객체의 배열을 리턴할 수도

```
class Person{
    String name;
    public Person(String name) { this.name = name; }
}
public class Test {
    public static void main(String[] args) {
        Person[] list = makeFriends(5);
        System.out.println(list[0].name);
    }
    private static Person[] makeFriends(int size) {
        Person[] list = new Person[size];
        for (int i = 0; i < list.length; i++) {
            list[i] = new Person("친구"+i);
        }
        return list;
    }
}
```

메소드의 활용과 FAQ

메소드 호출
매개변수 전달
리턴값
다양한 리턴 방식 (객체간 상호작용)

메소드 호출: 위치와 형태

- 어디에 있는 함수를 어떻게 호출하나요?
  - 같은 클래스 내의 다른 메소드: "메소드명()"
  - 다른 클래스의 static 메소드: "클래스명.메소드명()"
  - 다른 클래스의 일반 메소드: "객체변수.메소드명()"

```
public class Example {
    public static void main(String[] args) {
        // (1)클래스 내의 다른 함수 호출
        print();
        printChar('*', 10);
        int result = calculate(10,20,'-');
        System.out.println( calculate(10,20,'*') );
        // (2)다른 클래스 및 객체의 멤버 메소드 호출
        int positiveValue = Math.abs(-10);
        Person p = new Person();
        p.setName("홍길동"); // 객체의 기능 (method) 실행
        int size = p.length(); // 리턴값이 있는 함수 (예, 이름 글자수 구함)
    }
    private static void print() { /*코드생략*/ }
    private static void printChar(char c, int i) { /*코드생략*/ }
    private static int calculate(int i, int j, char c) { return 0; /*코드 생략*/ }
}
```

메소드 호출: 다른 메소드 호출

- 메소드 내에서 다른 메소드를 호출 할 수 있나요?
  - 물론입니다! 아래 예제를 살펴보세요.
  - Introduce() -> printName() // Person클래스 내부의 메소드호출
  - Introduce() -> Matching.evaluate() -> name.charAt() // 외부의 메소드 호출

```
class Matching {
   static int evaluate(String name) {
      return (name.charAt(0) - name.charAt(2));
   }
}
class Person{
   String name;
   void introduce() {
      printName();
      int diff = Matching.evaluate(name);
      if(diff==0) System.out.println("역순으로 같은이름");
   }
   void printName() { System.out.println("내이름="+name); }
}
```

- 처음 introduce()는 누가 실행하나요?
  - Person 객체를 생성한 곳(예, main()메소드)에서 실행 할 수 있겠죠
  - Person p = new Person();
  - p.introduce();

메소드 호출 더 알아보기 FAQ

- 생성자에서 메소드를 호출 할 수 있나요?
  - YES
  - 생성자에서 초기화 할 코드가 길어지거나 여러 생성자에서 공통으로 사용하는 함수가 필요한 경우 활용
  - 반대는 불가능. 즉, 메소드에서 생성자를 호출 할 수 없음
- 객체를 여러 개 만들면 객체들의 함수가 서로 겹치나요?
  - NO
  - 객체는 클래스(청사진, 설계도)로 만든 실체(instance)입니다. 여러 개 만들면 각각의 함수가 따로 실행되는 것입니다.
  - 물론 실행하는 알고리즘(로직)은 클래스의 설계도가 같기 때문에 동일하지만 처리하는 데이터(매개변수 혹은 멤버 변수의 값)가 다르면 결과도 다릅니다.
- 자기 자신을 호출할 수 있나요?
  - YES. 재귀 함수 (recursive method)라고 부르며, 자신이 계속 자신을 호출하게 됨
  - 무한 루프가 되지 않도록 종료 조건을 잘 설계해야 함
- 부모 클래스 메소드를 호출할 수 있나요?
  - YES. 상속받은 메소드는 내가 정의한 것과 동일한 효과.
  - 부모 클래스의 메소드를 재정의(오버라이딩)할 때도 부모 클래스 메소드를 호출 할 수 있으며 이때는 super.method명()로 호출 가능

생성자와 메소드 호출 비교

- 생성자의 호출 시점과 호출 방법
  - "new 클래스명()"으로 객체를 생성할 때 실행됨
    - 함수와 유사하게 동작하지만 return type을 선언하지 않음
  - 생성자에서 다른 생성자를 호출하는 경우도 있음
    - 같은 클래스 내의 다른 생성자 호출: this()
    - 생성자에서 부모 생성자 호출이 가능함: super()
- 생성자에서 다른 메소드 호출이 가능함
  - 반대로, 메소드에서는 생성자를 호출 할 수 없음
- 메소드처럼 오버로딩 가능
  - 매개변수의 개수나 타입이 다른 생성자를 여러 개 선언 가능

매개변수 전달 FAQ

- 함수가 종료되면 매개변수는 사라지나요?
  - 모든 매개변수는 메소드 내부에 선언한 로컬 변수가 됨
  - 함수가 호출되면 로컬 변수에 전달받은 값을 대입하는 구조
  - 이 로컬 변수는 메소드 종료 시에 모두 메모리에서 삭제됨

- 메소드의 매개변수 이름과 호출하는 쪽의 변수이 이름 같으면?
  - 무관합니다.
  - 예제에서 main()에도 변수 a가 있고, add()에도 변수 a가 별도의 지역 변수로 존재
  - 아래 add() 메소드에서 매개변수에 11이 전달됨 (로컬변수 int a = 11; 효과)
  - 오른쪽 리턴값이 있는 예제는 add(a+1)이 12로 치환되는 효과. (main()의 a값은 변함 없음)

```
public static void main(String[] args) {
    int a = 10;
    add(a+1);
    System.out.println(a); // 10 출력
}
private static void add(int a) {
    a++;
    System.out.println(a); // 12 출력
}
```

```
public static void main(String[] args) {
    int a = 10;
    add(a+1);
    System.out.println(a); // 10 출력
    System.out.println(add(a+1)); // 12 출력
}
private static int add(int a) {
    a++;
    return a;
}
```

매개변수 전달 FAQ

- 메소드 내에서 객체를 생성한 경우도 메소드가 종료될 때 객체가 메모리에서 사라지나요?
  - YES or NO
  - 가비지 콜렉터는 어떤 객체든 아무도 참조하지 않으면 삭제합니다. (레퍼런스하는 객체 변수가 없는 경우)
  - 메소드가 종료될 때 객체 변수를 리턴해서 호출하는 쪽에서 저장한다면 객체는 메모리에 남아 있게 됨
- 매개변수를 전달할 때 데이터의 타입이 항상 같아야 하나요?
  - YES or NO
  - 형변환(type casting)이 자동으로 되는 경우 다른 타입으로 전달 가능
    - int a 매개변수에 char 타입인 'A' 를 전달 가능
  - 객체의 경우 자식 클래스 객체를 부모 클래스로 전달 가능 (업캐스팅)
    - Student s = new Student(); 에서 Student가 Person 자식 클래스라면,
    - 변수 s를 void print(Person p){ } 메소드의 매개변수로 전달 가능
    - Person p = s; 이런 형태로 업캐스팅이 자동으로 이루어짐
    - 즉, print(new Student()); 이런 형태로 호출 가능

매개변수 – 가변 인자

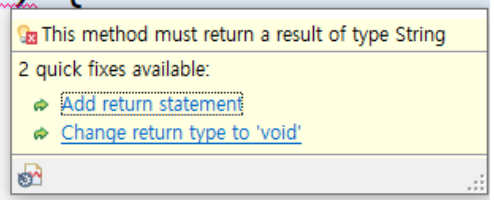
- 가변 인자 (variable arguments): 같은 타입이면서 알 수 없는 개수의 매개변수 처리가능
- 전달하는 쪽에서 굳이 배열을 만들지 않아도 됨
- 매개변수 선언 방식
  - typeName... parameterName

```
void printMax( int... numbers ) {
  //numbers가 배열처럼 동작함
}
```

리턴값 FAQ

- return을 했는데 오류가 납니다?
  - 모든 조건에서 return이 실행됨을 보장해야 하는데, 컴파일러는 조건문의 내용을 보지 않음
  - 아래 예제에서는 else return "보통"; 이렇게 하면 오류 없음

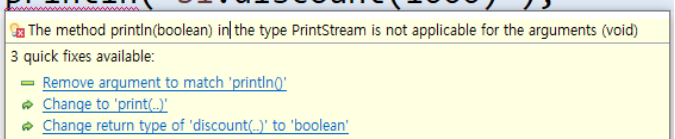
```
public class ReturnEx {
    public static void main(String[] args) {
        System.out.println( testScore(90) );
    }
    private static String testScore(int num) {
        if( num > 80 ) return "우수";
        else if( num <= 80 ) return "보통";
    }
}
```



리턴값 FAQ

- 메소드 결과를 출력하려는데 오류가 납니다?
  - 메소드 내에서 System.out.println()이 문자열을 리턴해주는 것이 아님에 주의
  - void 리턴 값을 함수의 매개변수로 전달할 수 없음

```
class Book{
    private String name;
    private int price;
    public Book(String n, int p){
        name = n;
        price = p;
    }
    String getName() {
        return name;
    }
    void discount(int dis) {
        price = price - dis;
        System.out.println("할인가격:" + price);
    }
}
public class Example2 {
    public static void main(String[] args) {
        Book b1 = new Book("Java 마스터", 20000);
        System.out.println( b1.getName() );
        System.out.println( b1.discount(1000) );
    }
}
```



리턴값 FAQ

- 메소드 뒤에 점(.)은 왜 쓰는 건가요?
  - 메소드의 리턴 값이 객체인 경우, 다시 객체를 참조하여 멤버 변수나 멤버 메소드 사용 가능
  - Person p = findPerson();
  - String name = findPerson().getName();
  - int size = findPerson().getName().length();

독특한 리턴방식(1)

- (1) 객체의 멤버 변수들을 업데이트 해두고, 나중에 가져가도록
  - 아래 예제에서, store()호출하면서 return을 받지 않고(못하고),
  - 나중에 필요한 정보를 객체 c를 참조해서 가져오는 방식

```
public class UpdateYou {
    public static void main(String[] args) {
        Computer c = new Computer();
        c.store("homework.hwp");
        System.out.println( c.getFileList() );
        System.out.println( c.getNumOfFiles() );
    }
}
```

독특한 리턴방식(2)

- (2) 매개변수에 자신의 객체의 레퍼런스를 전달하고, 상대방 메소드에서 나의 함수를 호출해서 필요한 값을 전달받는 방식
  - pc1.requestDownload(user1); 호출 시
  - pc1객체가 user1.onDownloadComplete() 메소드를 호출해줌 (값도 전달)

```
class PC{
    void requestDownload(User u) {
        // downloading...
        u.onDownloadComplete("a.mp3", 20);
    }
}
public class User{
    public static void main(String[] args) {
        User user1 = new User();
        PC pc1 = new PC();
        pc1.requestDownload(user1);
    }
    public void onDownloadComplete(String file, int time) {
        System.out.println("User는 다운로드한 파일 확인");
        System.out.println("이름=" + file + ", 재생시간:" + time );
    }
}
```

독특한 리턴방식(3)

- 매개변수 타입을 통일하기 위해서(규격화) 인터페이스를 사용
  - 앞의 예제에서는 User 클래스 타입을 설계 시점에 미리 알고 있어야 함
  - 추후 Person 클래스도 처리할 수 있도록 하려면 PC 클래스를 수정해야 함
  - 따라서, Person이 공통 인터페이스를 구현하도록 하면됨 (예제에서는 DownlaodWaiter 인터페이스 타입을 Person도 구현하면 됨)

```
class PC{
    void requestDownload(DownloadWaiter user){
        // downloading...
        user.onDownloadComplete("a.mp3", 20);
    }
}
interface DownloadWaiter {
    void onDownloadComplete(String file, int time);
}
public class User implements DownloadWaiter{
    public static void main(String[] args) {
        User user1 = new User();
        PC pc1 = new PC();
        pc1.requestDownload(user1);
    }
    @Override
    public void onDownloadComplete(String file, int time) {
        System.out.println("User는 다운로드한 파일 확인");
        System.out.println("이름=" + file + ", 재생시간:" + time );
    }
}
```

독특한 리턴방식 (static 변수와 메소드 공유)

- 공통 클래스 활용
  - static 멤버 변수, 멤버 메소드를 정의
  - 누구든 객체 레퍼런스 없이도 접근 가능
  - 따라서, 누구나 데이터를 기록해두고, 변경된 값을 가져갈 수 있음
  - 동기화 주의 필요. 공통 클래스 없이 한쪽 클래스에 static 변수/메소드 정의 가능

```
class Common {
    static int num;
    static String name;
}
class A{
    void write(int a){
        Common.num = a;
    }
}
class B{
    int read(){
        return Common.num;
    }
}
```

연습문제

- Word 클래스 만들고 다음의 기능(method)를 추가해 보세요.
  - 1. 단어에 포함된 알파벳 문자의 위치 찾기
  - 2. 단어 뒤집기
  - 3. 대소문자 반대로
  - 4. 문자(char)별로 몇 개 있는지 통계 출력