1. java.lang 패키지

1. Object

1. 자바에서 최상위 클래스, 모든 클래스들은 Object클래스를 상속받아 만들어짐.
2. protected Object clone() : 객체 자신을 복제하여 복제된 객체를 리턴.
3. public boolean equals(Object obj) : 객체자신과 obj가 같은 객체를 바라보고 있는지 확인하는 메소드.
4. public Class getClass() : 객체 자신의 정보를 담고있는 클래스를 리턴.
5. public String toString() : 객체 자신의 정보를 문자열로 변환하여 리턴.

2. String

1. String 변수명 = "문자열"; -> String 변수명 = new String("문자열");
2. String도 객체를 만들어서 사용하기 때문에 참조타입이다.
3. String은 immutable(불변성) 성질을 가지고 있다. 처음생성된 문자열(String)이 메모리에 저장되고 똑같은 변수에 다른 문자열을 대입했을 때 메모리에 저장되어 있는 값이 변경되는 게 아니고 새로운 문자열을 만들어서 메모리에 저장한다. <=> StringBuilder는 메모리에 저장되어 있는 문자열의 값을 변경한다.
4. String(String s) : 문자열을 받는 생성자
5. String(char[] chArr) : 문자배열을 받는 생성자
6. char charAt(int index) : 해당 인덱스의 char형 값 리턴.
7. int compareTo(String str) : String 변수; 변수.compareTo(Strint str) str과 비교해서 같으면 0, 사전순으로 변수의 값이 앞쪽이면 1, 뒤쪽이면 -1 리턴.
8. String concat(String str) : 매개변수로 받은 문자열 str을 뒤에 붙여 새로운 문자열 리턴.
9. boolean contains(String str) : str이 포함되어 있는지 아닌지 검사. 포함되어 있으면 true, 포함되어 있지 않으면 false 리턴.
10. boolean endWith(String str) : str로 끝나는지 아닌지 검사. true, false 리턴.
11. boolena equals(String str) : str과 문자열 같은지 다른지 비교.
12. int indexOf(char ch) : ch가 문자열에 몇번째 index에 존재하는 지 검사. 왼쪽부터 검사를 시작해서 처음만나는 ch의 index를 리턴. 못 찾으면 -1 리턴.
13. int indexOf(char ch, int pos) : pos부터 검사를 시작. 왼쪽부터 검사.

14. int indexOf(String str) : str이 존재하는 지 검사. str을 처음 만나는 인덱스를 리턴.
15. int lastIndexOf(char ch) : 오른쪽 검사를 시작해서 ch를 처음 발견한 위치를 리턴.(int lastIndexOf(String str))
16. int length() : 문자열의 길이를 리턴.
17. String replace(char old, char new) : 문자열에서 old를 찾아서 new바꿔서 새로운 문자열 리턴.
18. String replace(String old, String new) : 문자열에서 old를 찾아서 new바꿔서 새로운 문자열 리턴.
19. String replaceAll(String old, String new) : old에 해당하는 문자열을 찾아서 new로 모두 변경. 정규식표현식 사용가능.
20. String[] split(String regex) : 지정된 regex를 기준으로 분리하여 배열로 리턴. -> bit.naver.cloud split(".") => {bit, naver, cloud};
21. boolean startsWith(String str) : str로 시작하는 지 검사.
22. String substring(int begin) : begin 인덱스부터 끝까지 문자열 잘라서 리턴.
23. String substring(int begin, int end) : begin 인덱스부터 end 인덱스 전까지 문자열 잘라서 리턴.
24. String toLowerCase(), toUpperCase() : 영문자 소문자로 변환해서 리턴, 영문자 대문자로 변환해서 리턴.
25. String toString() : String 변수에 저장되어 있는 문자열 리턴.
26. String trim() : 문자열의 왼쪽 끝과 오른쪽 끝의 공백(길이와 상관없이 모두)을 제거한 새로운 문자열 리턴. 중간 공백들은 제거되지 않음
27. static String valueOf(다른타입 변수) : 변수에 저장되어 있는 값을 문자열로 변환하여 리턴.

3. StringBuffer

1. 버퍼 : 문자열 저장하고 편집하기 위한 공간.
2. StringBuffer는 String과는 다르게 메모리에 저장되어 있는 값을 직접 변경한다.
3. append() : String의 concat()과 같은 역할. 문자열을 합쳐준다. append() 문자열의 주소를 가져와서 그 주소에 담겨있는 문자열에 직접 작업을 한다.
4. StringBuffer equals() 오버라이딩 되어있지 않아서 두개의 StringBuffer 비교하려면 toString()를 사용해서 String으로 변환한 후 비교해야 된다.
5. StringBuffer의 생성
    - StringBuffer 변수명 = new StringBuffer(); //아무것도 지정하지 않으면 Buffer의 크기가 16으로 지정된다.
    - StringBuffer 변수명 = new StringBuffer(int 크기);
    - StringBuffer 변수명 = new StringBuffer(String str);
6. 메소드

- StringBuffer append(boolean, char, char[], double, float...) : 매개변수로 전달받은 값을 String으로 변환한 후 기존 값과 합쳐준다.
- int capacity() : 버퍼의 크기를 알려주는 메소드. length()와는 다르다.
- char charAt(int index) : index에 위치한 문자를 하나 꺼내서 리턴.
- StringBuffer delete(int startIdex, int endIndex) : startIndex ~ endIndex - 1까지의 문자열을 삭제한 후 StringBuffer를 리턴.
- StringBuffer delete(int index) : index 위치의 문자 하나 삭제.
- StringBuffer insert(int pos, 두번째인자(boolean, int, double, float, String, ...)) : 두 번째인자 값을 String으로 변환하여 pos 위치부터 삽입.
- int length() : 버퍼에 담겨있는 문자열의 길이 리턴.
- StringBuffer replace(int startIndex, int endIndex, String str) : startIndex ~ endIndex - 1까지의 문자열을 str로 변경.
- StringBuffer reverse() : 역순으로 정렬된 StringBuffer를 리턴.
- void setCharAt(int index, char ch) : index 위치의 문자를 ch로 변경.
- void setLength(int 새로운 문자열의 길이) : 문자열의 길이를 매개변수로 받아온 값으로 변경. 현재 문자열 길이보다 커지면 비어있는 곳은 공백(space)으로 채운다. 현재 문자열 길이보다 작아지면 남는 문자열은 삭제된다.
- String toString() : 문자열의 값을 String형태로 리턴.
- String substring(int index) : index 위치부터 문자열의 끝까지 잘라서 String 형태로 리턴.
- String substring(int startIndex, int endIndex) : startIndex~endIndex - 1까지 문자열 잘라서 String형태로 리턴.

4. StringBuilder

1. StringBuffer에 Thread개념을 뺀 클래스. Thread를 이용해서 값을 안전하게 동기화시키는 반면 StringBuilder는 Thread의 동기화 역할만 제거하여 성능을 향상시켰다.
2. StringBuilder 사용법
    - StringBuilder 변수명 = new StringBuilder();
3. StringBuilder는 StringBuffer와 사용법이 완전 동일하다.

5. Math

1. 수학적 계산이 필요한 코드에서 유용하게 사용할 수 있는 클래스.
2. Math클래스의 모든 메소드가 static이어서 객체생성없이도 사용할 수 있으며 Math클래스의 생성자 자체가 private이라 객체를 생성할 수도 없

다.

3. 올림, 반올림, 버림
    - ceil() : 소수점 자리의 값에 상관없이 올림
    - round() : 소수점 자리의 값이 5이상이면 올림 소수점 자리의 값이 5미만이면 버림
    - floor() : 소수점 자리의 값에 상관없이 버림
    - 10^n승을 이용하면 소수점 n자리까지 올림, 반올림, 버림을 할 수 있다.
    - (Double)Math.round(실수 * 10^n) / 10^n
    - Math.floor(실수 * 10 ^ n) / 10 ^ n
    - Math.ceil(실수 * 10 ^ n) / 10 ^ n
4. 절대값, 최대값, 최소값, 근사값
    - abs() : 절대값을 구해주는 메소드. 음수는 양수로 양수는 그대로.
    - max(숫자1, 숫자2) : 매개변수로 주어진 두 수중 큰 수를 리턴. 숫자1, 2의 타입이 동일해야 한다.
    - min(숫자1, 숫자2) : 매개변수로 주어진 두 수중 작은 수를 리턴. 숫자1, 2의 타입이 동일해야 한다.
    - rint(double 실수) : 매개변수로 주어진 실수에서 제일 가까운 정수를 double형으로 리턴. 1.5, 2.5, 3.5.. 등 중간 값은 짝수를 리턴.

6. 래퍼(Wrapper) 클래스

1. 원시타입 8가지는 객체가 아니다. 그러나 어쩔 수 없이 객체로 사용해야 하는 경우가 발생한다.
2. 래퍼클래스를 이용하면 원시타입 8가지에 대한 객체를 생성해서 사용할 수 있다.
3. char, int를 제외한 6가지는 타입이름의 첫글자만 대문자로 변경하면 래퍼클래스가 된다.
4. char => Character, int => Integer
5. Integer intNum = new Integer(100);
6. Integer와 함께 Number클래스를 상속받아 만들어진 클래스는 총 8가지이다. 래퍼클래스인 Byte, Short, Integer, Long, Float, Double 6가지 + BigInteger, BigDecimal
7. 박싱 : 원시타입의 변수를 래퍼클래스의 객체로 변환함
8. 언박싱 : 래퍼클래스의 객체를 원시타입의 값으로 변환
9. 컴파일러가 컴파일을 진행하다가 원시타입을 객체로 써야될 때는 자동 박싱을 하여 객체로 변환하여 사용하고 래퍼클래스의 객체를 원시타입으로 사용해야 될 경우에는 원시타입으로 자동 언박싱을 해서 사용한다.
10. 주로 사용하는 메소드
    - 래퍼클래스.valueOf(매개변수) : 매개변수 값을 래퍼클래스의 객체로 변환. 변환된 객체를 리턴.

- 래퍼클래스.parse원시타입(매개변수) : 매개변수 값을 원시타입으로 변환해서 리턴. 매개변수로는 String이나 CharacterSequence를 받는다.

  ```
  Integer.parseInt("100"); => 100이 int형으로 변환돼서
  ```

- 래퍼클래스의 객체.toString() : 객체에 담겨있는 값을 String형태로 변환해서 리턴.

  ```
  Integer.valueOf(100).toString();
  ```

2. java.util.regex 패키지

1. 정규표현식

1. 정규표현식이라는 것은 텍스트 데이터중 원하는 조건(패턴)과 일치하는 문자열을 찾아내기 위한 도구.
2. 미리 정의된 기호들과 문자들을 이용해서 패턴을 만든다.
3. Pattern 클래스
   - 정규표현식을 사용하게 되면 정규표현식이 Pattern클래스의 compile이라는 메소드를 통해 Pattern 객체로 만들어진다.
   - Pattern compile(String 정규표현식) : 매개변수로 받은 정규표현식을 Pattern객체로 만들어서 리턴.
   - 정규표현식으로 비교할 대상을 Pattern클래스의 matcher메소드를 통해 Matcher 객체로 만들어준다.
   - Matcher matcher(CharSequence 비교할 대상);
   - Mather객체에 있는 matches메소드를 이용해서 정규표현식에 부합하는지 검사한다.
   - boolean matches(); => 부합하면 true, 아니면 false

     ```
     String pattern = 정규표현식;

     //정규표현식에 해당하는 Pattern객체 생성
     Pattern pt = Pattern.complie(pattern);

     String compare = 비교할문자열;
     //비교를 위한 Matcher 객체 생성, 비교할 문자열을 매개변
     Matcher m = pt.matcher(compare);

     m.matches(); => compare가 pattern에 부합하면 true, 부
     ```

4. 정규표현식 패턴
    - c[a-z]* : 소문자 c로 시작하는 모든 영단어
    - c[a-z] : 소문자 c로 시작하는 두글자 영단어
    - c[a-zA-Z] : 소문자 c로 시작하고 대소문자 상관없는 두 글자 영단어.
    - c[a-zA-Z0-9] : 소문자 c로 시작하고 대소문자 상관없고 0~9까지 숫자도 가질 수 있는 두 글자 단어.
    - .* : 모든 문자열
    - c. : 소문자 c로 시작하는 두자리 문자열
    - c.* : 소문자 c로 시작하는 모든 문자열
    - c. : 소문자 c와 일치하는 모든 문자열. .는 이스케이프 문자
    - c[0-9] : 소문자 c로 시작하고 숫자와 조합된 두글자 단어.
    - c.*t : 소문자 c로 시작하고 소문자 t로 끝나는 모든 문자열
    - [b-c].* : 소문자 b 또는 c로 시작하는 모든 문자열
    - .c. : 소문자 c를 포함하는 모든 문자열
    - .*c.+ : 소문자 c를 포함하는 모든 문자열. c다음에 문자가 하나이상 존재해야 하고 c로 끝나는 문자열은 배제한다.
    - c.{2} : 소문자 c로 시작하는 세자리 문자열.

3. java.math 패키지

1. BigInteger

1. long으로 표현할 수 있는 자리수 19자리정도다. 이 이상되는 정수를 사용하고 싶을 때 사용할 수 있는 클래스가 BigInteger라는 클래스.
2. BigInteger객체 생성 방법
    - 생성자를 통해 객체를 생성
    - valueOf()를 통해 객체를 생성
3. 다른 타입으로 변환
    - toString() : 문자열로 변환
    - toByteArray() : 바이트배열로 변환
    - intValue() : int형으로 변환
    - longValue() : long형으로 변환
    - floatValue() : float형으로 변환
    - doubleValue() : double형으로 변환
4. BigInteger의 연산
    - BigInteger add(BigInteger val) : 덧셈연산
    - BigInteger subtract(BigInteger val) : 뺄셈연산
    - BigInteger multiply(BigInteger val) : 곱셈연산
    - BigInteger divide(BigInteger val) : 나눗셈연산(몫)
    - BigInteger remainder(BigInteger val) : 나머지연산

2.BigDecimal

1. double 타입은 범위가 넓지만 정밀도가 13자리 밖에 되지 않아서 실수형 오차를 표현하는 것을 정밀하게 할 수 없다.
2. BigDecimal 정밀도를 정수*10^-scale까지 표현가능하고 scale 0~Integer.MAX_VALUE(대략 21억) 정밀도를 10^-21억까지 표현할 수 있어서 실수형 오차를 매우 정밀하게 표현할 수 있다.
3. BigDecimal의 객체 생성
    - 생성자를 통한 객체 생성
    - valueOf()를 통한 객체 생성
    - 생성자나 valueOf() 매개변수로 double형 값을 넣어주면 실수형에서 오차가 발생할 수도 있다.

      `new Decimal(1.11) => Deciamal의 값이 1.1100000000000`

4. 다른 타입으로 변환
    - intValue() : int형으로 변환
    - longValue() : long형으로 변환
    - floatValue() : float형으로 변환
    - doubleValue() : double형으로 변환
5. BigDecimal의 연산
    - BigDecimal add(BigDecimal val) : 덧셈연산
    - BigDecimal subtract(BigDecimal val) : 뺄셈연산
    - BigDecimal multiply(BigDecimal val) : 곱셈연산
    - BigDecimal divide(BigDecimal val) : 나눗셈연산(몫)
    - BigDecimal remainder(BigDecimal val) : 나머지연산
6. divide(), setScale()
    - divide(BigDecimal val, MathContext mc: 반올림모드를 가지고 있는 클래스)
    - MathContext클래스의 RoundingMode
        - HALF_UP : 5이상 올림, 5미만 버림(반올림)
        - HALF_EVEN : 반올림(반올림 자리 값이 짝수면 6이상 올림 6미만 버림, 홀수면 5이상 올림 5미만 버림)
        - HALF_DOWN : 6이상 올림 6미만 버림
        - CEILING : 올림
        - FLOOR : 버림
        - UP : 양수일 때는 올림, 음수일 때는 버림
        - DOWN : 양수일 때는 내림, 음수일 때는 올림
    - setScale(int newScale) : 표현할 소수점 자리수를 newScale로 변경. 오차가 발생할 수 있어서 RoundingMode를 지정해서 반올림을 어떻게 진행할 지 정해줘야 한다.