クラス定義とオブジェクトの生成・使用

5-2

Main.java

Compile error

```
1  import java.io.BufferedReader;
2  import java.io.InputStreamReader;
3
4  public class Main {
5    final int a;
6    Main(int b){ a=b; }
7    void method(int c){a=c;}
8    public static void main(String[]
9        Main t = new Main(10);
10       t.method(20);
11       System.out.println(t.a);
12     }
13 }
14
```

Final修飾子が指定されている

コンパイルエラー

実行 (Ctrl-Enter)

コンパイルエラー　入力　コメント 0

```
Main.java:7: error: cannot assign a value to final variable a
  void method(int c){a=c;}
                     ^
1 error
```

5-4

5-5

Main.java

```java
import java.io.BufferedReader;
import java.io.InputStreamRea

public class Main {
    void x(int num){
        num +=30;
    }
    public static void main(String[] args){
     int num =10;
     Main t = new Main();
     t.x(num);
     System.out.println(num);
    }
}
```

基本データ型

実行 (Ctrl-Enter)

出力 入力 コメント 0

```
10
```

Main.java

```java
import java.io.BufferedReader;
import java.io.InputStreamReader

public class Main {
    void x(int[] num){
        num[0] +=30;
    }
    public static void main(String[] args){
     int[] num ={10};
     Main t = new Main();
     t.x(num);
     System.out.println(num[0]);
    }
}
```

参照型

実行 (Ctrl-Enter)

出力 入力 コメント 0

```
40
```

Main.java ✖ Test.java ✖ ＋

```
1  import java.io.BufferedReader;
2  import java.io.InputStreamReader;
3
4  class Test{
5      static void methodA(){
6          methodB();
7          Test.methodB();
8          methodC();
9          Test.methodD();
10     }
11     static void methodB()
12     void methodC(){
13     methodB();
14     Test.methodB();
15     methodD();
16     Test.methodD();
17     }
18     void methodD()
19 }
20
21
```

実行 (Ctrl-Enter)

Staticメソッド→インスタンスメソッド

Staticメソッド→インスタンスメソッド(クラス名での呼び出し)

インスタンスメソッド→インスタンスメソッド(クラス名での呼び出し)

クラス内で定義したインスタンスメンバは、クラス内で定義したstaticメンバに直接アクセスできる

クラス内で定義したstaticメンバは、クラス内で定義したインスタンスメンバに直接アクセスできない。アクセスする場合は、インスタンス化してからアクセスする

Staticメソッドは「クラス名.メソッド名」という呼び出しは許可されているが、インスタンスメンバは許可されていない

Main.java

```java
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Main {
    public static void main(String[] args)  {
        // Your code here!
        Main ojb1 = new Main();
        Main ojb2 = new Main();
        System.out.print(ojb1 == ojb2);
        ojb2 = operate(ojb1,ojb2);
        System.out.print(" "+(ojb1 == ojb2));
    }

    static Main operate(Main ojb1,Main ojb2){
        Main ojb3 = ojb1;
        ojb1 = ojb2;
        return ojb3;
    }
}

```

実行 (Ctrl-Enter)

出力 入力 コメント 0

```
false true
```

参照先が同じオブジェクトであれば
trueを返し、異なる場合はfalseを返す

Operate内は「引数を二つ取り、最初の引数の中身を返す」動きをしているので

```
ojb2 = operate(ojb1, ojb2);
ojb2 = ojb1;
```

異なるパッケージにあるクラスを利用する

①import文でインポートする

A.Import com.se.myprj.users;

B.import com.se.myprj.*;

クラス名に代えて「*」を書くと指定されたパッケージに属するすべてのクラスをインポートできる

②完全名で指定する

D.com.se.myprj.User user = new com.se.myprj.User();

インポートされていないクラスを利用するには、パッケージ名を含めた完全な名前で指定する

5-18　カプセル化のメリット

- カプセル化とはクラス（インスタンス）を一つのオブジェクトとしての独立性を高めようとするもの
- クラス（インスタンス）に対するアクセス制御を行う

↓

予期せぬバグを防いだり、クラス（インスタンス）の変更や組換えなどが容易になる

D.オブジェクトが保持するデータを保護できる

5-18　カプセル化のメリット2

C.不変オブジェクトを設計できる

不変オブジェクトの条件

- オブジェクトの内部状態を変更可能なメソッドを提供しない
- どのメソッドもオーバーライドされないことを保証する(サブクラスからの状態の変更を防ぐ)
- 全てのフィールドを"private final"で定義する("final"の指定は必須ではないが、より明示的に不変クラスであることを宣言できる)
- 内部に可変オブジェクトを保持している場合、そのオブジェクトを外部に提供しない。また、該当可変オブジェクトの値を内部的に変更しない