Hiding Information

Recall: the class *fraction*

```
class fraction
{
    int num;
    int den;
……….
}
```

- the fields num and den are directly **available to be read and altered from any other class in any package**
- Therefore in our main method, we can access these fields like so:

```
fraction f = new fraction();
f.num=4;
f.den=0;
```

However……

- When creating classes, we may not want to give direct access of the fields (a.k.a variables) from any point outside the class.

Why?

- We want our objects to stay in a certain form by restricting direct access to the fields.

- The process of creating programs so that information within a class is made inaccessible from outside the class is called encapsulation

Syntax: controlling access

- By adding *private* to the declarations of the fields, the field will only be accessible to the instance methods in the class.

- e.g.

```
class fraction
{
    private int num;
    private int den;
……….
}
```

- Now we can no longer access the fields directly from another class like so:

```
public static void main (String[] args){
    fraction f = new fraction();
    f.num=4;
    f.den=5;
}
```

- Now we can only access the fields through the instance methods that were created for this class.
  - **Hence, we are controlling the manipulation of these objects**

- Because we can no longer access fields directly, we must use instance methods to access these fields.
- If we need to access specific fields from a class, we can employ several strategies

1) accessor methods

- Are methods that return a field from a class.

2) mutator methods

- are methods that alter the value of a field in a class.

- Create an instance method called *putnum* that will receive a parameter *x* of type int. This method will set the *num* field in the object to *x*.
- Create an instance method called *getnum* that will return the num field in the object.

```
public class fraction {
	private int num;
	int den;


	//*****************************************************************
	// putnum sets the num field to x.
	public void putnum(int x){
		num = x;
	}

	// getnum retruns the num field
	public int getnum(){
		return num;
	}
}
```

For example:

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

      fraction f = new fraction();

      f.putnum(7);
      f.den = 9;

      System.out.println(f.getnum());
      System.out.println(f.den);
}
```

- When calling an instance method:
  - we create an object called fraction
  - Using that object, we invoke the instance method.
  - **Notice we now do not have to directly access the field num. We access this field through our instance methods**

Accessing two objects from the same instance method

Try This…..

- Create a instance method called *comparef*. This method will receive a parameter of type fraction stored in a variable called *temp*. When called, this method will compare the objects decimal (I.e. n/d) value to the decimal value of temp and return the larger value.

- To create an instance method that can access another object, we will send the object as a parameter.

public fraction comparef (fraction temp){

}

- The preceding header for the instance method:
  - Is named '*comparef*'; and will be a part of *fraction* (implicit object)
  - Receives a parameter of type fraction and is stored in a variable called 'temp'; (explicit object)
  - Returns a fraction;

- NOTE: Because *comparef* is an instance method belonging to one object and another object is being passed in, we have access to two objects!

- The private fields belonging to the object being passed in are called explicit fields. You will need to use accessor and mutator methods to access these fields. (e.g. temp.getnum(), temp.putnum()).

- The fields belonging to the object that the instance method was called from are called implicit fields. You can access these directly (e.g. num)

Here is the complete method

```
public fraction comparef (fraction temp){

    fraction larger = new fraction();
    if ((num/den)>(temp.getnum()/temp.den)){
        larger.putnum(num);
        larger.den = den;
    }
    else {
        larger.putnum(temp.getnum());
        larger.den=temp.den;
    }
    return (larger);
}
```

Notice:

- **the fields from the implicit object is referred to as num and den while**
- **public fields from the explicit objects are referred to as temp.den or larger.den.**
- **Private fields from the explicit objects are accessed via accessor and mutator methods**

Calling this instance method

In your main program….

```
public static void main(String[] args) {
    fraction f = new fraction();
    f.putnum(2);
    f.den = 4;
    fraction g = new fraction();
    g.putnum (1);
    g.den = 3;
    fraction h;

    h=g.comparef(f);
}
```

Note: the object g is the implicit object calling the instance method. The object f is being passed as a parameter. The result is stored in object h.