Methods – accessors and mutators

Introduction to OO Programming

Encapsulation

- Encapsulation is an important principle of OO Programming.
- Encapsulation describes the ability of an object to hide its data and methods from the rest of the world.
- This allows an object to control how its data is manipulated.
- It is important that instance variables can only be accessed and manipulated through methods provided in the class.
- The java keywords `public` and `private` are an important aspect of this.

Protecting instance variables

- The access modifier `private` is usually applied to instance variables.
- This makes sure that they can only be accessed by code within the same class – the methods.
- Making an instance variable `public` means that it can be accessed and modified by code from other classes.
- This is usually not good practice.
- Make your instance variables `private`, and allow access via `public` methods.

`public` instance variables

File Student.java

```
public class Student
{

// instance variables
public int age;

// methods









}
```

File StudentTest.java

```
public class StudentTest
{

public static void main(String[] args)
{

Student myStudent = new Student();

myStudent.age = -21;









}
```

`private` instance variables

File Student.java

```
public class Student
{

// instance variables
private int age;

// methods
public int getAge()
{
   return age;
}



}
```

File StudentTest.java

```
public class StudentTest
{

public static void main(String[] args)
{

Student myStudent = new Student();

myStudent.age = -21;
// syntax error - age has
// private access in Student.java


}
```

`private` instance variables

File BankAccount.java

```
public class BankAccount
{

// instance variables
private double balance = 0;

// methods
public double getBalance()
{
   return balance;
}



}
```

File BankTester.java

```
public class BankTester
{

public static void main(String[] args)
{

BankAccount myAcc = new BankAccount();

myAcc.balance = 1000000.00;
// syntax error - balance has
// private access in BankAccount.java


}
```

`private` instance variables

File BankAccount.java

```
public class BankAccount
{
// instance variables
private double balance = 0;

// methods
 public void withdraw(double amount)
  {
      if(balance >= amount)
      {
         balance = balance - w;
      }
      else
      {
         System.out.print("Insufficient funds");
      }
   }
}
```

Accessors

- If you make your instance variables private, you generally need some way of accessing them. Best practice is to do this via `public` methods.
- Methods provided to retrieve the values of instance variables are known as *Accessors*. Accessors are used to get the value of particular instance variables without changing that value.
- Accessors are also known as Getters.
- For example, a `BankAccount` class might have a `getBalance( )` accessor method. This would allow **appropriate** access to the instance variable.

Accessor return type

- Accessors are used to retrieve the value of particular instance variables.
- Therefore, their return type is the type of the instance variable in question.
- For example, a **`getBalance( )`** method would return an item of type double, if the instance variable **`balance`** was a double.
- A **`getAccountNumber( )`** method would return an item of type int, if the instance variable **`accountNumber`** was in int.
- Because accessors do not modify the values of instance variables, they do not require input parameters.

Using an Accessor method

File BankAccount.java

```
public class BankAccount
{

// instance variables
private String name = new
String();

// methods
public String getName()
{
   return name;
}


}
```

File BankTester.java

```
public class BankTester
{

public static void main(String[] args)
{

BankAccount acc1 = new BankAccount();
String accountHolder = new String();
accountHolder = acc1.getName();

System.out.print(accountHolder);


}
```

Using an Accessor method

File BankAccount.java

```
public class BankAccount
{

// instance variables
private String name = new
String();

// methods
public String getName()
{
   return name;
}


}
```

File BankTester.java

```
public class BankTester
{

public static void main(String[] args)
{

BankAccount acc1 = new BankAccount();



System.out.print(acc1.getName());


}
```

Mutators

- Sometimes you want to provide public methods to allow code to directly change the value of private instance variables.
- Methods provided to directly modify the instance variables of an object are known as *Mutators*. They are used to modify the values of one or more instance variables in an appropriate way.
- Mutators are also known as Setters.
- For example, a **`BankAccount`** class might have a **`setAccountName( )`** mutator method.

Mutator input parameters

- Mutators will require at least one input parameter.
- The type of input parameter will match the type of the instance variable being set.
- For example, a **`setName( )`** mutator method will require an input parameter of type String if the instance variable **`name`** is a String
- A **`setNumber( )`** mutator method will require an input parameter of type int if the instance variable **`number`** is an int.
- Mutators do not return anything, so their return type is void.

Using a mutator method

File BankAccount.java

```
public class BankAccount
{

// instance variables
private String accountName =
new String();

// methods
public void
setAccountName(String name)
{
  accountName = name;
}

}
```

File BankTester.java

```
public class BankTester
{

public static void main(String[] args)
{

BankAccount acc1 = new BankAccount();

acc1.setAccountName("John");




}
```

`private` methods

- It's possible to make a method `private`. This means it can only be called by code within the same class.
- This is useful in certain situations – for example if you want to provide code for use in other methods.
- Most commonly used when you want to break tasks into smaller tasks but don't want the smaller tasks to be accessed by code outside the class. When the smaller tasks don't make sense except as part of the larger task.
- You will encounter `private` methods in second year programming.

`private` methods

File Student.java

```
public class Student
{

// instance variables
private int age;

// methods
private int getAge()
{
   return age;
}



}
```

File StudentTest.java

```
public class StudentTest
{

public static void main(String[] args)
{

Student myStudent = new Student();

int age = myStudent.getAge();
// syntax error - method getAge() has
// private access in Student.java



}
```

Summary

- Encapsulation describes the ability of an object to hide its data and methods from the rest of the world. Instance variables are usually given `private` access. Methods are usually given `public` access.
- You generally need to provide `public` methods to access `private` instance variables.
- Accessor methods are used to retrieve the values of instance variables without changing them. Accessor methods are very common.
- Mutator methods are used to set the values of instance variables. Mutator methods are not as common as accessor methods.