



# LETS LEARN JAVA

## ENCAPSULATION

# Training Plan - Basics

---

## Topics to be covered

What is encapsulation  
Concept of packages

Access Modifiers

Beans

---

# Encapsulation

Encapsulation is a method of controlling access of attributes and methods of a class from outside the class.

Encapsulation in Java is achieved using two features

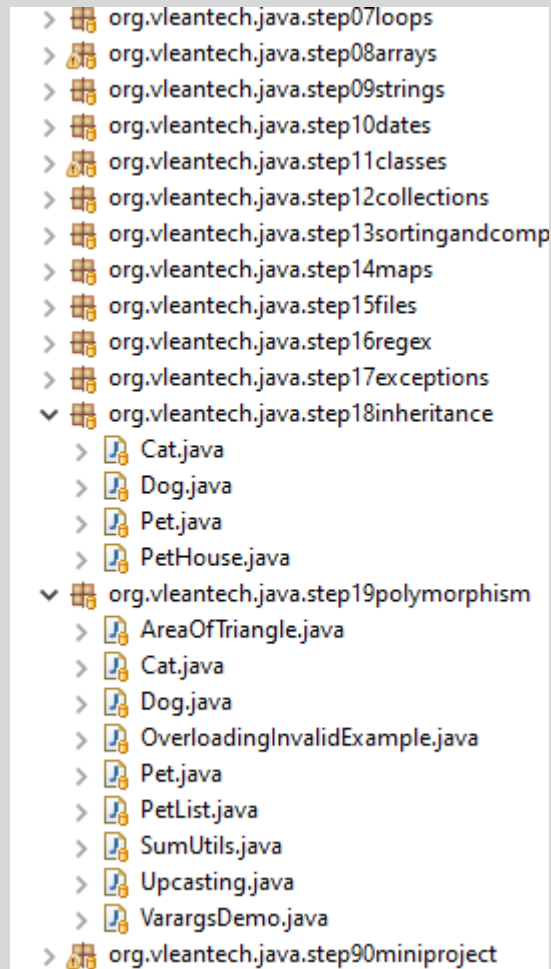
1. Packages
2. Access Modifiers

# Packages

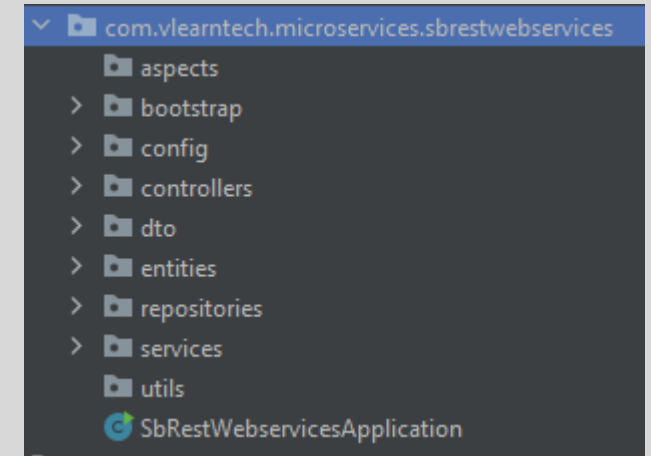
Package is a group or folder of classes, usually related

Mapping classes under packages allow you to:

1. Reuse same class names under a different context
2. Organize your code better
3. Provide additional protection through access modifiers



```
> org.vleantech.java.step07loops
> org.vleantech.java.step08arrays
> org.vleantech.java.step09strings
> org.vleantech.java.step10dates
> org.vleantech.java.step11classes
> org.vleantech.java.step12collections
> org.vleantech.java.step13sortingandcomp
> org.vleantech.java.step14maps
> org.vleantech.java.step15files
> org.vleantech.java.step16regex
> org.vleantech.java.step17exceptions
▼ org.vleantech.java.step18inheritance
  > Cat.java
  > Dog.java
  > Pet.java
  > PetHouse.java
▼ org.vleantech.java.step19polymorphism
  > AreaOfTriangle.java
  > Cat.java
  > Dog.java
  > OverloadingInvalidExample.java
  > Pet.java
  > PetList.java
  > SumUtils.java
  > Upcasting.java
  > VarargsDemo.java
> org.vleantech.java.step90miniproject
```



```
▼ com.vleantech.microservices.sbrestwebservices
  > aspects
  > bootstrap
  > config
  > controllers
  > dto
  > entities
  > repositories
  > services
  > utils
  SbRestWebservicesApplication
```

# Packages

```
package org.vleantech.java.step20encapsulation;
```

```
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
import java.util.Random;
```

```
import org.vleantech.java.step19polymorphism.Cat;  
import org.vleantech.java.step19polymorphism.Dog;  
import org.vleantech.java.step19polymorphism.Pet;
```

```
public class PetList {  
    public static void main(String[] args) {  
        List<Pet> pets = new ArrayList<>();  
        pets.add(new Dog("Bruno", "Brown", 3, "German Shepherd"));  
        pets.add(new Dog("Tiny", "Black", 4, "Labrador"));  
        pets.add(new Dog("Spooky Spider", "Striped", 2, "Golden Retriever"));  
        pets.add(new Cat("Tim", "Gray", 3));  
        pets.add(new Cat("Silky", "White", 2));  
    }  
}
```

To indicate a class is part of a specific package, we use the package keyword on top of the class.

To access a class from another package, there are three options available.

1. Import specific class (**recommended**)
2. Import all classes of the package
3. Instead of importing, use the full named reference of the class being used.

Option 1 has been shown in the screenshot where specifically Pet, Dog and Cat classes from another package have been imported for use.

# Packages – Alternate ways of access Classes

```
package org.vleantech.java.step20encapsulation;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

import org.vleantech.java.step19polymorphism.*;

public class PetList {
    public static void main(String[] args) {
        List<Pet> pets = new ArrayList<>();
        pets.add(new Dog("Bruno", "Brown", 3, "German Shepherd"));
        pets.add(new Dog("Tiny", "Black", 4, "Labrador"));
        pets.add(new Dog("Spooky Spider", "Striped", 2, "Golden Retriever"));
        pets.add(new org.vleantech.java.step18inheritance.Cat("Tim", "Gray", 3));
        pets.add(new org.vleantech.java.step18inheritance.Cat("Silky", "White", 2));
    }
}
```

**Option 2** – Using package.\* to import all classes from the package.

**Option 3** – Avoid import and directly provide fully qualified name of the class being used

# Access Modifiers

As the name states, this is used to define/modify the access levels for attributes, and methods of any class. It may even be used with classes.

Access Modifier	Within Class	Within Package another class	Outside Package but inherited sub classes	Outside Package and non inherited classes
<b>private</b>	Yes	No	No	No
<b>default</b>	Yes	Yes	No	No
<b>protected</b>	Yes	Yes	Yes	No
<b>public</b>	Yes	Yes	Yes	Yes

Let us see a few examples of usage of these access modifiers

# Bean

```
public class Employee {  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private Character gender;  
    private BigDecimal age;  
    private LocalDate joiningDate;  
    private Integer salary;  
    private String phone;  
    private Region region;  
}
```

## In a bean:

1. All attributes are private
2. All attribute retrievals are done using get and set methods, also known as getters and setters

```
public Long getId() {  
    return id;  
}  
  
public String getFirstName() {  
    return firstName;  
}  
  
public String getLastName() {  
    return lastName;  
}  
  
public Character getGender() {  
    return gender;  
}  
  
public BigDecimal getAge() {  
    return age;  
}  
  
public LocalDate getJoiningDate() {  
    return joiningDate;  
}  
  
public Integer getSalary() {  
    return salary;  
}  
  
public String getPhone() {  
    return phone;  
}  
  
public Region getRegion() {  
    return region;  
}
```

```
public void setId(Long id) {  
    this.id = id;  
}  
  
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
  
public void setLastName(String lastName) {  
    this.lastName = lastName;  
}  
  
public void setGender(Character gender) {  
    this.gender = gender;  
}  
  
public void setAge(BigDecimal age) {  
    this.age = age;  
}  
  
public void setJoiningDate(LocalDate joiningDate) {  
    this.joiningDate = joiningDate;  
}  
  
public void setSalary(Integer salary) {  
    this.salary = salary;  
}  
  
public void setPhone(String phone) {  
    this.phone = phone;  
}  
  
public void setRegion(Region region) {  
    this.region = region;  
}
```

A bean with only getter methods is a **read only bean**. That means once constructed, no one can update any value. This is seen from time to time, like an immutable bean to read data from a file.

Similarly a bean with only setter methods is a **write only bean**. There are rare.