1. What is Encapsulation in Java? Why is it called Data hiding?
Ans:
In simple language, encapsulation is the process of encapsulating something. The definition for encapsulation in Java is the binding of data and corresponding methods into a single unit is called "Encapsulation".
Encapsulation consists of two subrules, Data Hiding, and Data Abstraction.
Data Hiding means that we are hiding the data from the world or public view. For example, in terms of Object Oriented Programming, a TV remote class has its Data Hidden. We have various keys in the remote that perform necessary functions like changing the channel or increasing/decreasing the volume. We just know that on pressing certain keys, certain actions will be triggered without knowing the internal working or data involved with it. The internal data and working are **hidden** from public view, i.e, hidden from the world. This process is called Data Hiding, where we are only showing the necessary data(like keys) and hiding(not exposing) the internal data and working from the public. Since for Encapsulation, Data Hiding is a primary rule, encapsulation is also called Data Hiding.
Data Abstraction is somewhat similar to Data Hiding in the sense that we are only exposing data to the public that is exposable and hiding the remaining as the knowledge of the internal data and working is not necessary for the end user. In other words, Data Abstraction is the reduction of a particular body of data to a simplified representation of the whole. Abstraction, in general, is the process of removing characteristics from something to reduce it to a set of essential elements.
In Java, if a class follows Data Hiding and Data Abstraction then it is referred to as an Encapsulated class.

2. What are the important features of Encapsulation?
Ans:
Data Hiding and Data Abstraction are the important features of Encapsulation.
Data Hiding means that we are hiding the data from the world or public view. For example, in terms of Object Oriented Programming, a TV remote class has its Data Hidden. We have various keys in the remote that perform necessary functions like changing the channel or increasing/decreasing the volume. We just know that on pressing certain keys, certain actions will be triggered without knowing the internal working or data involved with it. The internal data and working are **hidden** from public view, i.e, hidden from the world. This process is called Data Hiding, where we are only showing the necessary data(like keys) and hiding(not exposing) the internal data and working from the public.

3. What are getter and setter methods in Java? Explain with an example.
Ans:
Setter methods are used to set the value to private instance variables of the class.
Syntax for the setter method:
a. compulsory the method name should start with set.
b. it should be public.
c. return type should be void.
d. compulsorily it should have some argument.

Example:
rollNumber is a private data member of a class Student

```
private int rollNumber;
```

In order to set a value to this variable outside the class Student, we need to add a public setter method that takes an integer argument rollNumber from the user as follows:

```
public void setRollNumber(int rollNumber){
    this.rollNumber = rollNumber;
}
```

Here, we used this keyword to indicate that it is an object variable and not a local variable. The return type should be void as it does not return anything.
By calling this setter method outside the Student class, we can set the roll number to the Student object.

Getter methods are used to get the value from private instance variables of the class.
Syntax for the getter method:
a. compulsory the method name should start with get.
b. it should be public.
c. return type should not be void.
d. compulsorily it should not have any argument.

Example:
rollNumber is a private data member of a class Student

```
private int rollNumber;
```

In order to get a value from this variable outside the class Student, we need to add a public getter method that takes no argument but returns the roll number of a Student object as follows:

```
public int getRollNumber(){
    return rollNumber;
}
```

It should return an integer roll number so we added the return type as int, and it should return the roll number when getRollNumber() method is called.
By calling this setter method outside the Student class, we can get the roll number of the Student object.

4. What is the use of this keyword? Explain with an example.
Ans:
“this” keyword helps in removing the Shadowing problem faced in setter methods that have the same instance variable name as the local variable name. If both the local variable and instance variable have the same name inside the method then it would result in a name clash and JVM will always give preference for the local variable.
For example

```java
class Student {
private String name;
private Integer id;
private String address;

public void setStudentData(String name, integer id, String address){
    name = name;
    id = id;
    address = address;
}
```

In the above code, there is a setter method which takes name, id and address as input for an object and sets those values for the object. So, when we create an object and pass these arguments as input and then try to print it,

```java
Student s1 = new Student();
s1.setStudentData("Rahul", 22, "Bengaluru");
System.out.println(s1.name + " " + s1.id + " " + s1.address);
```

We get the output as: `null 0 null`
This is because once the objects are created, default values are assigned to it and these default values are printed. Even though we passed different arguments which are not default values, JVM(or Java Compiler/IDE) gives preference to the local variable in the setter method and the local variables currently hold default values, hence the output with default values. JVM gives priority to local variables during name clash in the setter method.

“this” keyword holds a reference to the current object and by adding it to the LHS of the assignment operator in the setter method for the variables, the local variable value that is passed as an argument is added to the object’s variable now.
Whichever object is calling the setter method “this” refers to the address of that object.
Example:

```java
class Student {
private String name;
private Integer id;
private String address;

public void setStudentData(String name, integer id, String address){
    this.name = name;
    this.id = id;
    this.address = address;
}
```

Now, if we use this method on an object and try to print the values, we get the desired result:

```
Student s1 = new Student();
s1.setStudentData("Rahul", 22, "Bengaluru");
System.out.println(s1.name + " " + s1.id + " " + s1.address);
```

We get the output as: `Rahul 22 Bengaluru`

5. What is the advantage of Encapsulation?
Ans:
Some of the advantages of encapsulation are:
a. It improves the security of data, as we are not exposing the entire data to the public.
b. Enhancement becomes easy, if we want to add some new data or work in the codebase then enhancing it would be easier as we don't need to share the entire details with the public, just sharing what the enhanced feature does is enough.
c. Maintainability and modularisation are easier. Maintaining the codebase becomes efficient as the code is having many modules for different work.
d. It provides flexibility to the user to use the system very easily.

6. How to achieve encapsulation in Java? Give an example.
Ans:
Encapsulation in Java can be achieved by making every data member inside the class private and access to this private data we need to use public getter and setter methods. By using private modifiers we can implement "data hiding". In the code below, we have followed the rules for making an encapsulated class, so the class Student is encapsulated.
Example:
We create a student class where each student has two properties name and roll number and any student can set/get his/her name and roll number using public setters and getters. Getters and Setters are made public so that people can access these values outside the class.

```
class Student{
    private String name;
    private int rollNumber;
    // Above, we have made both the variables(data members) as private
    // public method to set the name
    public void setName(String name) {
        this.name = name;
    }
    // public method to get the name
    public String getName() {
        return name;
    }
    // public method to set the roll number
    public void setRollNumber(int rollNumber) {
        this.rollNumber = rollNumber;
    }
```

```java
    // public method to get the roll number
    public int getRollNumber() {
        return rollNumber;
    }
}

public class Test {
    public static void main(String[] args){
        Student s1 = new Student();
        s1.setName("Rahul");
        String s1Name = s1.getName();
        Student s2 = new Student();
        s1.setName("PW");
        String s2Name = s2.getName();
        s1.setRollNumber(1159);
        s2.setRollNumber(2203);
        int s1RollNumber = s1.getRollNumber();
        int s2RollNumber = s2.getRollNumber();
        System.out.println("The details of student s1 are " + "Name: "+
s1Name + "and Roll Number: "+s1RollNumber);
        System.out.println("The details of student s2 are " + "Name: "+
s2Name + "and Roll Number: "+s2RollNumber);
    }
}
```