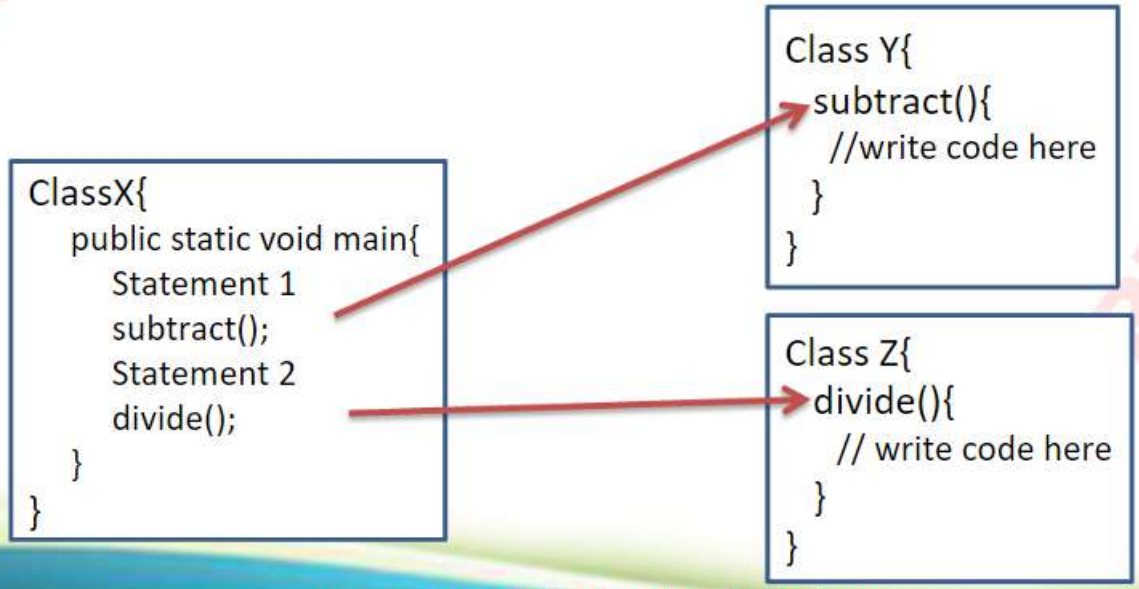
Java Methods

Click to Continue

- **Java method**s are set of statements to perform a specific operation which can be added inside a java class.
- This set of statements can be invoked by other methods in the same(or) different class by using the method name.

Illustration: Main method in class X invokes method *"subtract"* in class Y and method "*divide*" in class Z.



Method Declaration

Click to Continue

Syntax:

```
<specifier> <returnType> <methodName>(<parameter>*) {
        <statement>*
}
```

Example:

```
public int add(int x, int y){
        int sum = x+y;
        return sum;
}
```

Returning values from method

Click to Continue

return is a keyword used to return values from a method.

Syntax:

return <return Value>;

Where,

- <return Value> is the variable whose value needs to be returned.
- The "return Value" data type must be same as the one specified in the method signature..
- **Returning control:** If you need to stop the method execution and pass the control back to the calling method use the return without returning any value.

  Syntax: return;

Multiple Return from a method

Click to Continue

- A method can have any number of return statements
- You can also return constants from methods.

Below is an example of a code having multiple return statements

```
package com.accessspecifier;

public class ReturncDemo {

    public int calculateSalary(int age) {
        int salary = 0;

        if (age > 18) {
            salary = 10000 + 5000;
            return salary;
        } else {
            return 5000;
        }

    }
}
```

Returning a variable which holds a value

Returning a constant value this could be any constant boolean, long , string etc,

NOTE: It is NOT a good practice to return multiple times in a method. This results in poor maintainability and readability of the code.

How to avoid multiple return; Click to Continue

```
package com.accessspecifier;

public class ReturnGoodDemo {

    public int calculateSalary(int age) {
        int salary = 0;

        if (age > 18) {
            salary = 10000 + 5000;

        } else {
            salary = 5000;
        }

        return salary;
    }
}
```

Value stored in a variable and returned once.

Try It out – Methods And Solution

Click to Continue

Develop the following class,

1. Create a Java class "Circle.java" with a method "calculateCircumference"
2. This method should accept radius as argument, calculate the circumference and return the circumference.
3. The main method should invoke the Circle objects "calculateCircumference" method by passing a value for the length, say 10.
4. The main method should also print the circumference (result returned by the calculateCircumference method).

Try It out – Methods And Solution

Click to Continue

Develop the following class,

1. Create a Java class “Circle.java” with a method “calculateCircumference”
2. This method should accept radius as argument, calculate the circumference and return the circumference.
3. The main method should invoke the Circle objects “calculateCircumference” method by passing a value for the length, say 10.
4. The main method should also print the circumference (result returned by the calculateCircumference method).

```
package com.accessspecifier;

public class Circle {

    public int calculateCircumference(int radius) {
        float circumference = 0;
        circumference = 2 * 3.14f * radius;
        return circumference;
    }

    public static void main(String args[]) {
        Circle c = new Circle();
        float circumference = c.calculateCircumference(10);
        System.out.println("The circumference of circle is " + circumference);
    }
}
```

What and How of Encapsulation?

Click to Continue

Encapsulation is a OOP concepts it is also called "***Data Hiding***" which,

- Is a protective barrier which protects the data in a class from being directly accessed by the code outside the class.
- Access to the data is controlled by defining a wrapper method which process the variable.

How is a class variable Encapsulated?

- The fields in a class are made **private** so that it cannot be accessed by any other class.
- The encapsulated fields can be **accessed** only by using the **public methods** specified in the class.
- Encapsulated data is accessed using the "**Accessor (getter)**" and "**Mutator (setter)**" methods.
  - **Accessors** – Methods to retrieve the hidden/encapsulated data.
  - **Mutators** – Methods to change hidden /encapsulated data

Application of encapsulation.

Click to Continue

Encapsulation is used in developing **transfer objects** or **value objects** in applications. These objects act as a bundle to carry forward data from presentation to back end tiers.

Illustration:

- Assume you are trying to register in a social media site by entering name and address.
- **RegistrationVO** – this will be the transfer object with encapsulated fields **name** & **address.**
- The data you enter will be placed inside **RegistrationVO** sent to the server to be processed.

Try it out – Transfer object

Click to Continue

Assume that you are developing a registration screen where you need to capture the fields name,DOB, email , phone. Develop a transfer object *RegistrationVO* with respective fields and getters and setters.

NOTE: Follow naming convention for attribute and methods and also ensure that they are meaningful.

Try it out – Transfer object

Click to Continue

```java
import java.util.Date;

public class RegistrationVO {

    private String name;
    private String email;
    private String phone;
    private Date dateOfBirth;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }
```

Getters and setters to access the encapsulated fields.

Try it out – Transfer object

Click to Continue

```java
import java.util.Date;

public class RegistrationVO {

    private String name;
    private String email;
    private String phone;
    private Date dateOfBirth;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }
```

Getters and setters to access the encapsulated fields.

How do you create the object and set the desired values

```java
package com.accessspecifier;

import java.util.Date;

public class Encapsulation {

    public void populateRegistrationVO() {
        RegistrationVO vo = new RegistrationVO();

        vo.setEmail("abc@gmail.com");
        vo.setName("Jack");
        vo.setPhone("9883391919");
        vo.setDateOfBirth(new Date("14/07/2012"));


        Date dob = vo.getDateOfBirth();
        String email = vo.getEmail();
        String name = vo.getName();
        String phone = vo.getPhone();

    }
}
```

The data is set using mutators

The values are retrieved using accessors.

Method Overloading

Click to Continue

Method overloading is two different versions of the same method available in the same class.

How is method overloading done?

This is done by either,

- Changing the input parameter type
- Changing the number of parameters.

Illustration:

```
package com.accessspecifier;

public class Overload {

    public float calculateCircumference(int radius) {
        float circumference = 0;
        circumference = 2 * 3.14f * radius;
        return circumference;
    }


    public float calculateCircumference(float radius) {
        float circumference = 0;
        circumference = 2 * 3.14f * radius;
        return circumference;
    }


    public float calculateCircumference(int radius, float pieValue) {
        float circumference = 0;
        circumference = 2 * pieValue * radius;
        return circumference;
    }
}
```

Data type changed.

Number of parameters changed.

Try It out & Solution - Method Overloading

Click to Continue

Create a calculator class and implement a method *add.*

Method 1: add(int a, int b) – adds two integers and returns a integer.

Method 2: add(float a, float b) – adds two integers and returns a float.

Method 3: add(int a, int b, int c) – adds three integers and returns the sum.

Invoke the three methods from the main method and print the returned value.

Try It out & Solution - Method Overloading

Click to Continue

Create a calculator class and implement a method *add.*

Method 1: add(int a, int b) – adds two integers and returns a integer.

Method 2: add(float a, float b) – adds two integers and returns a float.

Method 3: add(int a, int b, int c) – adds three integers and returns the sum.

Invoke the three methods from the main method and print the returned value.

Solution

```
public class Calculator {

    public int add(int a, int b) {
        int c = a + b;
        return c;
    }

    public float add(float a, float b) {
        float c = a + b;
        return c;
    }

    public int add(int a, int b, int c) {
        int d = a + b + c;
        return d;
    }

    public static void main(String[] args) {
        int result;
        float resultFloat;
        Calculator c = new Calculator();
        result = c.add(10, 20);
        System.out.println(result);
        resultFloat = c.add(10.5f, 20.4f);
        System.out.println(resultFloat);
        result = c.add(10, 20, 30);
        System.out.println(result);
    }
}
```

Access modifier- Static

Click to Continue

What is static?

Static keyword is used with method or variable. Let us look at what they means in the next subsequent slides.

Illustration:

```
variable: private static int age = 0;
method: public static calculateSalary(){
            // some code here
        }
```

Static variables are variables which are globally available across all the instances of an object.

Illustration: Assume there is a class Employee with a member variable "*String company*" declared as static. Assume we create two instances of the employee object *E1 and E2* . If in one instance the company name set as "*Ripples*". IF we try to print the company name of *E2* this will print *Ripples* this is because the company is static and shared by all the Employee objects.

Salient Points:

- If an object is declared static there will only be only one instance of that object.
- Static method or variable of an object can be referenced without creating an object instance.

Try It Out – Static variable

Click to Continue

Static variable – Let us develop the following code to understand how static behaves.

```
package com.accessspecifier;

public class StaticDemo {

    static int num1 = 100;
    int num2 = 200;

    public void changeValue(int value1, int value2) {
        num1 = value1;
        num2 = value2;

    }

    public static void main(String args[]) {
        StaticDemo c1 = new StaticDemo();
        c1.changeValue(300, 400);
        StaticDemo c2 = new StaticDemo();
        System.out.println("Value of C1 num 1--->" + c1.num1);
        System.out.println("Value of C1 num 2--->" + c1.num2);
        System.out.println("Value of C2 num 1--->" + c2.num1);
        System.out.println("Value of C2 num 2--->" + c2.num2);
    }
}
```

Output

```
Value of C1 num 1--->300
Value of C1 num 2--->400
Value of C2 num 1--->300
Value of C2 num 2--->200
```

Reason:

Since num1 is defined as static the variable value will be shared by both the objects C1 & C2. So though C1 initialize num1 it will be reflected in C2 also.

Static method

Click to Continue

*"**static**"* method of a class can be invoked without creating an instance of the class, so it can be invoked using the class name as reference.

```
package com.accessspecifier;

public class StaticMethodDemo {

    public static int num1 = 100;

    public static void changeValue(int value1) {
        num1 = value1;
    }

    public static void main(String args[]) {
        StaticMethodDemo.changeValue(200);
        StaticMethodDemo.num1 =300;
    }
}
```

Variables accessed using Class name.