**Working with classes and interfaces:**

Overview:

Declaring classes

Class members

Working with objects

Encapsulation and access modifiers

Field accessors and mutators

Declaring classes:

Java is Object oriented Language

Object encapsulate data, operations and usage semantics from how is done

Allow storage and manipulation details to be hidden

Separates what is to be done

A class is a template of Creating Objects

Declaring a class keyword followed by the class name

Body of the class is contained within brackets

Java source file name normally has same name as class

Class Flight{

// class members

}

Classes:

A class is made up of both state and executable code

Fields: Store object state

Methods:

Executable code that manipulates state and perform operations

Constructors:

Executable code used during object creation to set initial state

Class Flight {

Int passengers;

Int seats

}

//does not have a return type, same name as that of the class

Flight ()

{

Seats=150;

passengers =0

}

Void add1Passenger ()

{

If(passengers<seats)

{

Passengers+=1;

}

}

package com.org.workingwithclasses.classes;  
  
public class MathEquation {  
  
 double leftValue;  
 double rightValue;  
 char opsCode;  
 double result;  
  
 void execute() {  
 switch (opsCode) {  
 case 'a':  
 result = leftValue + rightValue;  
 break;  
 case 's':  
 result = leftValue - rightValue;  
 break;  
 case 'm':  
 result = leftValue \* rightValue;  
 break;  
 case 'd':  
 if (rightValue != 0) {  
 result = leftValue / rightValue;  
 } else {  
 System.*out*.println("given number val2 is zero");  
 }  
 break;  
 default:  
 System.*out*.println("invalid input");  
 result = 0.0d;  
 }  
 }  
  
  
}

Using classes:  
  
use the new keyword to create a class instance (aka object)

Flight flight;

flight= new Flight();

* Allocates memory described by the class , and runs the constructor
* Return a memory to the allocated memory

Understanding classes and reference types:

package com.org.workingwithclasses.classes;  
  
public class MathEquation {  
  
 public double leftValue;  
 public double rightValue;  
 public char opsCode;  
 public double result;  
  
 public void execute() {  
 switch (opsCode) {  
 case 'a':  
 result = leftValue + rightValue;  
 break;  
 case 's':  
 result = leftValue - rightValue;  
 break;  
 case 'm':  
 result = leftValue \* rightValue;  
 break;  
 case 'd':  
 if (rightValue != 0) {  
 result = leftValue / rightValue;  
 } else {  
 System.*out*.println("given number val2 is zero");  
 }  
 break;  
 default:  
 System.*out*.println("invalid input");  
 result = 0.0d;  
 }  
 }  
  
  
}

package com.org.workingwithclasses.classes;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 MathEquation[] equations = new MathEquation[4];  
 equations[0] = *create*(100.0d, 50.0d, 'd');  
 equations[1] = *create*(25.0d, 92.0d, 'a');  
 equations[2] = *create*(22.0d, 17.0d, 's');  
 equations[3] = *create*(11.0d, 3.0d, 'm');  
  
 for (MathEquation equation : equations) {  
 equation.execute();  
 System.*out*.println("result=" + equation.result);  
 }  
 }  
  
 private static MathEquation create(double leftVal, double rightValue, char opsCode) {  
 MathEquation mathEquation = new MathEquation();  
 mathEquation.leftValue = leftVal;  
 mathEquation.rightValue = rightValue;  
 mathEquation.opsCode = opsCode;  
 return mathEquation;  
 }

**Encapsulation and Access Modifiers:**

The implementation details of a class are generally hidden.

The concept is known as encapsulation. Java uses access modifiers to achieve encapsulation.

Basic access modifiers:

|  |  |  |  |
| --- | --- | --- | --- |
| Modifier | Visibility | Usage on classes | usage on members |
| NO access modifier | Only with in the same package | Y | Y |
| public | Every where | Y | Y |
| private | Only within the decaring class | N | Y |
|  |  |  |  |

As private applies to top-level classes; private is available nested classes.

package com.org.workingwithclasses.accessmodifiers;  
  
 class Flights {  
  
 int passengers;  
 int seats;  
  
 Flights()  
 {  
 System.*out*.println("Flight constucter is created");  
 }  
  
 void addPassenger()  
 {  
 System.*out*.println("add passenger");  
 }  
}

package com.org.workingwithclasses.accessmodifiers;  
  
public class RunningMainSamePackage {  
  
 public static void main(String[] args) {  
 Flights flights = new Flights();  
  
 flights.addPassenger();  
  
  
 }  
  
}

Running outside of the package giving compilation issues as there is no public access modifier.

package com.org.workingwithclasses;  
  
import com.org.workingwithclasses.accessmodifiers.Flights;  
  
public class Outsidepackage {  
 public static void main(String[] args) {  
 Flights flights = new Flights();  
 flights.addPassenger();  
 }  
}

Special References: this and null

This :

Implicit reference to current object

Useful for reducing ambiguity allows an object to pass itself as parameter.

package com.org.workingwithclasses.accessmodifiers;  
  
public class ThisKeywordnullkey {  
  
 int a = 10;  
 int b = 20;  
  
 public void m1(int x, int y) {  
 a = a + x;  
 b = b + y;  
  
 System.*out*.println(a);  
 System.*out*.println(b);  
 }  
  
}

package com.org.workingwithclasses.accessmodifiers;  
  
public class Specialreference {  
  
 //  
 // implicit reference to the current object: this  
 int a = 10;  
 int b = 20;  
  
 public void m1(int a, int b) {  
 this.a = this.a + a;  
 this.b = this.b + b;  
  
 System.*out*.println(this.a);  
 System.*out*.println(this.b);  
 }  
  
}

Null;

Represent an uncreated object

package com.org.workingwithclasses.accessmodifiers;  
  
public class RunningMainSamePackage {  
  
 public static void main(String[] args) {  
 Flights flights1 = new Flights();  
 Flights flights2 = new Flights();  
  
 Flights flights3 = null;  
  
 if (flights1.hasRoom(flights2)) {  
 flights3 = new Flights();  
 }  
  
 System.*out*.println(flights3);  
 }  
  
}

Filed accesses and mutators  
Setters and Getters

The specific fields a class uses to manage data values is generally considered to be an implementation detail

IN most cases these details should not be directly accessible outside of the worked

Use methods to control field access

package com.org.workingwithclasses.accessmodifiers;  
  
public class Employee {  
  
 private int age;  
  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
 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;  
 }  
  
 private String name;  
 private String email;  
  
  
}

package com.org.workingwithclasses.accessmodifiers;  
  
public class EmployeeMain {  
  
 public static void main(String[] args) {  
 //this is not right practice  
 Employee employee = new Employee();  
  
 employee.setAge(40);  
 employee.setName("virat");  
 employee.setEmail("test@gmail.com");  
 int age = employee.getAge();  
 String email = employee.getEmail();  
 String name = employee.getName();  
 System.*out*.println("Employee age is +"+age+" email id is "+email+" name is "+name);  
  
 }  
}

Summary:

Class creation: Template for creating objects

Object Creation: Instance of class

Classes are reference types

-class variables simply hold a reference

Instances created with new keyword

Multiple variables can reference same instance

Fields:

Stores object state

Methods:

-Executable code

-Manipulate state

- Perform operations

Constructor:

* Executable code
* Runs during object Creation
* Sets initial state

This keyword:

Implicit reference to current object

Null: represent un created object

Access modifiers:

Control class visibility

Control member visibility

Enable encapsulation

Setters & getters

Accessors mutators:

Use method to provide access

Accessors(getters) retrieve field values

05/01/2025:

Default initial state of fields

Field initializers

Constructor

Chaining Constructor

initialization blocks

Initialization and construction order

Class Initial State:

When an object is created, it is expected to be in useful state

Default initial state set by java often not enough

May need specific action

Set fields values directly, execute code

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| byte short int long | float double | char | boolean | Reference Types |
| 0 | 0.0 | ‘\u0000' | FALSE | null |

Three ways establish initial state:

1)Filed initializers: Specify fields initial value as part of the field’s declaration

-Can be equation

-can include other fields

-can include method calls

Public class earth {

Long circumreferenceInmiles=24901;

Long circumreferenceinkms= (long)(24901\*1.6d)

}

public class Earth {  
 long circumreferenceInmiles = 24901;  
 long circumreferenceinkms = m1();  
 public long m1() {  
 return 1;  
 }  
  
}

2) Constructors

package com.org.workingwithclasses.constructors;  
  
public class Person {  
 int panId;  
 String name;  
 char gender;  
 boolean marritalStatus;  
 double salary;  
  
 public Person()  
 {  
 System.*out*.println("person constructor");  
 panId=15658;  
 name="rajesh";  
 gender='M';  
 marritalStatus=true;  
 salary=568.25;  
 }  
}

Must have at least one

When no explicit constructor, java provides one

Can have multiple

Each must hava a unique parameters list  
Different number of parameters

Different parameters types

package com.org.workingwithclasses.constructors;  
  
public class Person {  
 int panId;  
 String name;  
 char gender;  
 boolean marritalStatus;  
 double salary;  
  
 //it is not taking any parameters  
 public Person() {  
  
 }  
 //it is taking parameters  
 public Person(int panId, String name, char gender, boolean marritalStatus, double salary) {  
 this.panId = panId;  
 this.name = name;  
 this.gender = gender;  
 this.marritalStatus = marritalStatus;  
 this.salary = salary;  
 }  
  
  
// public Person()  
// {  
// System.out.println("person constructor");  
// panId=15658;  
// name="rajesh";  
// gender='M';  
// marritalStatus=true;  
// salary=568.25;  
// }  
  
  
}

package com.org.workingwithclasses.constructors;  
  
public class PersonMain {  
  
 public static void main(String[] args) {  
  
 Person person=new Person(890,"name",'F',false,8585.5);  
  
 System.*out*.println(person.panId);  
 System.*out*.println(person.marritalStatus);  
 System.*out*.println(person.gender);  
 System.*out*.println(person.name);  
 System.*out*.println(person.salary);  
  
 }  
}

Constructor chaining:

One constructor can call another constructor

Use this keyword followed by parameter list

package com.org.workingwithclasses.constructors;  
  
public class Person {  
 int panId;  
 String name;  
 char gender;  
 boolean marritalStatus;  
 double salary;  
  
 //it is not taking any parameters  
 public Person() {  
  
 }  
  
 public Person(int panId, String name) {  
 this.panId = panId;  
 this.name = name;  
 }  
  
  
 //it is taking parameters  
 public Person(int panId, String name, char gender, boolean marritalStatus, double salary) {  
 this(panId,name);  
 this.gender = gender;  
 this.marritalStatus = marritalStatus;  
 this.salary = salary;  
 }  
  
  
}

Constructor Visibility:

If constructor is private we can not create a object.

package com.org.workingwithclasses.constructors;  
  
public class Person {  
 int panId;  
 String name;  
 char gender;  
 boolean marritalStatus;  
 double salary;  
  
 //it is not taking any parameters  
 public Person() {  
  
 }  
  
 public Person(int panId, String name) {  
 this.panId = panId;  
 this.name = name;  
 }  
  
  
 //it is taking parameters  
 private Person(int panId, String name, char gender, boolean marritalStatus, double salary) {  
 this(panId,name);  
 this.gender = gender;  
 this.marritalStatus = marritalStatus;  
 this.salary = salary;  
 }  
}

CalEngine:

package com.org.workingwithclasses.constructors;  
  
public class MathEquation {  
  
 private double leftValue;  
 private double rightValue;  
 private char opsCode;  
 private double result;  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public char getOpsCode() {  
 return opsCode;  
 }  
  
 public void setOpsCode(char opsCode) {  
 this.opsCode = opsCode;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public MathEquation() {  
  
 }  
  
 public MathEquation(char opsCode) {  
 this.opsCode = opsCode;  
 }  
  
 public MathEquation(double leftValue, double rightValue,char opsCode) {  
 this(opsCode);  
 this.leftValue = leftValue;  
 this.rightValue = rightValue;  
 }  
  
 void execute() {  
 switch (opsCode) {  
 case 'a':  
 result = leftValue + rightValue;  
 break;  
 case 's':  
 result = leftValue - rightValue;  
 break;  
 case 'm':  
 result = leftValue \* rightValue;  
 break;  
 case 'd':  
 if (rightValue != 0) {  
 result = leftValue / rightValue;  
 } else {  
 System.*out*.println("given number val2 is zero");  
 }  
 break;  
 default:  
 System.*out*.println("invalid input");  
 result = 0.0d;  
 }  
 }  
  
  
}

package com.org.workingwithclasses.constructors;  
  
public class Main {  
  
 public static void main(String[] args) {  
 MathEquation[] equations = new MathEquation[4];  
 equations[0] = new MathEquation(100.0d, 50.0d, 'd');  
 equations[1] = new MathEquation(25.0d, 92.0d, 'a');  
 equations[2] = new MathEquation(22.0d, 17.0d, 's');  
 equations[3] = new MathEquation(11.0d, 3.0d, 'm');  
  
 for (MathEquation equation : equations) {  
 equation.execute();  
 System.*out*.println("result=" + equation.getResult());  
 }  
 }  
}

Initialization blocks:

Share code across all constructors

* Cannot receive parameters
* Place code with brackets outside of any method or constructor

A class can have multiple

* All always execute
* Execute in order starting at the top of the source file

package com.org.workingwithclasses.constructors;  
  
public class Flight {  
  
public Flight()  
{  
 System.*out*.println("constructor call");  
}  
 private int passengers;  
 int seats=1;  
 private int flightNumber;  
 private char flightClass;  
 private boolean[] isSeatAvailable=new boolean[seats];  
  
 public int getPassengers() {  
 return passengers;  
 }  
  
 public int getSeats() {  
 return seats;  
 }  
  
 public int getFlightNumber() {  
 return flightNumber;  
 }  
  
 public char getFlightClass() {  
 return flightClass;  
 }  
  
 public boolean[] getIsSeatAvailable() {  
 return isSeatAvailable;  
 }  
  
// this is iniliation block  
 {  
  
 System.*out*.println("inilization block");  
 passengers=90;  
 seats=100;  
 flightClass='B';  
 flightNumber=98999;  
 isSeatAvailable[0]=false;  
 }  
}

Order:

Field initializers

Initialization blocks

Constructor

Summary:

Object initial state

-initial state expected to be useful

-java provides default field values

Field initializers

-set initial value as part of declaration

-can include an equation, other fields, and method calls

Constructer:

* Code can run during the object creation
* Accept zero or more parameters
* Can have multiple

One constructor can call another constructer:

* Call must be first line of constructor
* Can pass parameters

Constructer can be non public:

* Limits which code can perform specific types of instance creation
* Inilization blocks:

**02/05/2024**

Using static Members:

Static Fields

Static methods

Static import

Static initialization blocks

Static members are shared class-wide

* Not associated with individual instance

Declared using the static keyword

* Accessible using the class name

Static Members:

Static Field: A value not associated with a specific instance

All instances access the same value

package com.org.workingwithclasses.staticmembers;  
  
public class Employee {  
 static String *orgName*;  
 public static void m1Static()  
 {  
 System.*out*.println(*orgName*);  
 }  
}

package com.org.workingwithclasses.staticmembers;  
  
public class EMployeeMain {  
  
 public static void main(String[] args) {  
  
 Employee.*m1Static*();  
  
 }  
  
}

Calengine static:  
  
package com.org.workingwithclasses.staticmembers;  
  
public class MathEquation {  
  
 private double leftValue;  
 private double rightValue;  
 private char opsCode;  
 private double result;  
  
 private static int *numberOfcalculations*;  
 private static double *sumOfResult*;  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public char getOpsCode() {  
 return opsCode;  
 }  
  
 public void setOpsCode(char opsCode) {  
 this.opsCode = opsCode;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public MathEquation() {  
  
 }  
  
 public MathEquation(char opsCode) {  
 this.opsCode = opsCode;  
 }  
  
 public MathEquation(double leftValue, double rightValue, char opsCode) {  
 this(opsCode);  
 this.leftValue = leftValue;  
 this.rightValue = rightValue;  
 }  
  
 public void execute() {  
 switch (opsCode) {  
 case 'a':  
 result = leftValue + rightValue;  
 break;  
 case 's':  
 result = leftValue - rightValue;  
 break;  
 case 'm':  
 result = leftValue \* rightValue;  
 break;  
 case 'd':  
 if (rightValue != 0) {  
 result = leftValue / rightValue;  
 } else {  
 System.*out*.println("given number val2 is zero");  
 }  
 break;  
 default:  
 System.*out*.println("invalid input");  
 result = 0.0d;  
 }  
 *numberOfcalculations*++;  
 *sumOfResult* = *sumOfResult* + result;  
 }  
  
 public static double getAverage() {  
 return *sumOfResult* / *numberOfcalculations*;  
 }  
  
  
}

package com.org.workingwithclasses.staticmembers;  
  
  
public class Main {  
  
 public static void main(String[] args) {  
 MathEquation[] equations = new MathEquation[4];  
 equations[0] = new MathEquation(100.0d, 50.0d, 'd');  
 equations[1] = new MathEquation(25.0d, 92.0d, 'a');  
 equations[2] = new MathEquation(22.0d, 17.0d, 's');  
 equations[3] = new MathEquation(11.0d, 3.0d, 'm');  
  
 for (MathEquation equation : equations) {  
 equation.execute();  
 System.*out*.println("result=" + equation.getResult());  
 }  
  
  
 double average = MathEquation.*getAverage*();  
 System.*out*.println("average=" + average);  
  
 }  
}

**Static import:**

package com.org.workingwithclasses.staticmembers;  
import static java.lang.Math.\*;  
public class StaicImportExample {  
  
 public static void main(String[] args) {  
 System.*out*.println(*sqrt*(4));  
 System.*out*.println(*pow*(2, 2));  
 System.*out*.println(*abs*(6.3));  
 }  
}

**Static initialization:**

package com.org.workingwithclasses.staticmembers;  
  
public class Staicinilization {  
 private int id;  
 private String name;  
  
 public Staicinilization() {  
 System.*out*.println("default constructor");  
 id = 10;  
 name = "kk";  
 System.*out*.println(id);  
 System.*out*.println(name);  
  
 }  
 {  
 System.*out*.println("initilizatoion block");  
 System.*out*.println(id);  
 System.*out*.println(name);  
 }  
  
 static {  
 System.*out*.println("static initilization block");  
 }  
}

package com.org.workingwithclasses.staticmembers;  
  
public class StaticInilizationBlock {  
  
 public static void main(String[] args) {  
Staicinilization staicinilization=new Staicinilization();  
  
  
 }  
}

Static members

* Shared class wide
* Declared using static keyword

Static fields:

* Values not associated with an instance
* All instances access the same value

Static method:

* Perform action not tied to instance
* Can only access static members

Static inilization blocks:

* Perform one-time initialization
* Execute before types first use

Static vs nonstate Observation:

package com.org.workingwithclasses.staticmembers;  
  
public class StaticCase {  
  
 static int *value*;  
  
 public StaticCase()  
 {  
 *value*++;  
 }  
  
}

package com.org.workingwithclasses.staticmembers;  
  
public class StaticCaseExample {  
 public static void main(String[] args) {  
 StaticCase staticCase1=new StaticCase();  
 StaticCase staticCase2=new StaticCase();  
 System.*out*.println(staticCase1.*value*);  
 System.*out*.println(staticCase1.*value*);  
 System.*out*.println(StaticCase.*value*);  
 }  
}

package com.org.workingwithclasses.staticmembers;  
  
public class NonStatic {  
  
 int value;  
  
 public NonStatic()  
 {  
  
 }  
  
 public NonStatic(int value)  
 {  
 this.value=value;  
 }  
}

package com.org.workingwithclasses.staticmembers;  
  
public class NonStaticMain {  
  
 public static void main(String[] args) {  
  
 NonStatic nonStatic1 = new NonStatic(10);  
 NonStatic nonStatic2 = new NonStatic(20);  
  
 System.*out*.println(nonStatic1.value);  
 System.*out*.println(nonStatic2.value);  
 }  
  
  
}

Passby Value and passby reference:  
  
package com.org.workingwithclasses.passingObejectsasparameters;  
  
import java.awt.\*;  
  
public class Flight {  
 public int getFlightNumber() {  
 return flightNumber;  
 }  
  
 public void setFlightNumber(int flightNumber) {  
 this.flightNumber = flightNumber;  
 }  
  
 int flightNumber;  
  
 public Flight() {  
  
 }  
  
 public Flight(int flightno) {  
 flightNumber = flightno;  
 }  
  
  
 public static void swapFlight(Flight i, Flight j) {  
 int k = i.getFlightNumber();  
 i.setFlightNumber(j.getFlightNumber());  
 j.setFlightNumber(k);  
 }  
}

package com.org.workingwithclasses.passingObejectsasparameters;  
  
public class FligtMaoin {  
  
 public static void main(String[] args) {  
  
 Flight f1 = new Flight(10);  
 Flight f2 = new Flight(20);  
  
 System.*out*.println(f1.flightNumber);  
 System.*out*.println(f2.flightNumber);  
  
  
 Flight.*swapFlight*(f1,f2);  
  
 System.*out*.println(f1.flightNumber);  
 System.*out*.println(f2.flightNumber);  
  
 }  
}