② Difference between static and final method and field in Java

Feature	Static	final
Defination	Belongs to the class, not instances	Prevents modification (for variables) or overriding (for methods)
Fields	Shared across all instance of the class.	cannot be reassigned after initialization
Methods	Can be called using the class name	Cannot be overridden in a subclass.
Inheritance	Not inherited (cannot be) overriden	Cannot be inherited but not overriden
Modification	Can be modified (unless also final)	Cannot be modified after assignment .
Usage	Used when data/ methods should be shared across instances	Used when immutabilit or method restrictio is required .

In Java , you can access static fields and methods using an object reference , but it is not recommended because static members belong to the class rathere than any specific instance .

```
Class Example {
Static int staticvar = 10 ;
int $instancevar = 20 ;
Static void static method ( ) {
System . out . Println (" Static method Called ");
}
}
Public class Test {
Public static void main (String[] args) {
Example obj = new Example ( ) ;
System . out . Println (" Accessing Static var Via
    Object : " + obj . staticVar ) ;
system.out.Println (" Accessing static via class:"
                          + Example.staticVar);

  obj.staticMethod();

  Example.staticMethod();

  }
  }
```

⊙ Access Modifiers in Java is a keyword that defines the visibility (accessibility) of class, methods, variables, and constructors.

It determines how other classes and objects can access the members of a class. Java Provides four access modifiers.

1. <u>Public</u> :- The member is accessable from everywh

2. <u>Private</u> :- The member is accessable only within the class in which it is defined.

3. <u>Protected</u> :- The member is accessable within the same Package and by subclasses (even if they are in different packages)

4. <u>Default</u> : If no access modifier is specifie it is considered Package-Private. The memb is accessible only within the same Package

Comparison of Accessability

Access Modifier	Description	Same class	Same Package	Sub Classes	Different Packages
Public	The member is accessible from any class	Yes	Yes	Yes	Yes
Private	Any member is accessible only within its own class	Yes	No	No	No
Protected	The member is accessible within the same Package by the subclasses	Yes	Yes	Yes	Yes (if subclass)
Default (no modifier)	The member is accessible only within the same Package	Yes	Yes	No	No

Java supports three different types of variable based on their scope, memory allocation, and lifetime. The three types variable are

1. Local variable.
2. Instance variable.
3. static Variable,

1. Local variable

- Declare inside a method, constructor or block.
- Access only within the method or block.
- No default value is assigned ; must be initialize before use.

```
Public
Class LocalVarExample {
Void show() {
int localvar = 10 ;
System.out.println ("Local variable :" +localvar)
}
}
```

2. Instance variable

- Declare inside a class but outside any method.
- Belongs to an object and is initialized when the object is created.
- Has a default value (0 for numbers, null for objects, false or boolean).

```
Class Instancevar Example{
int instancevar =50 ;
void display( ){
system. out. Println(" Instance variable :" +instancevar)
}
}
```

Static variable :- (class variable)

- Declared using the static keyward insid a class but outside any method.

- shared among all instance of the class.

- initialized only once at class loading ti

- Has a default value.

```
class staticVarExample {
static int staticVar = 100 ;
Static void display () {
system. out. Println (" static variable : " +staticvar);
}
}
```

⑧ write a Program that can determine the letter, whitespace, and digit.

```
Import java.util.Scanner;
Public class CharacterTypeChecker{
Public Static void main (String [] args){
Scanner Sc~~anner~~ = new Scanner (system.in);

System.out.Print (" Enter a String: ");
String input = Sc.nextLine();

for (int i=0; i< input.length(); i++){
Char ch = input.CharAt(i);

if (Character.isLetter(ch))
  System.out.println (ch+ "is a Letter.");
else if (Character.ISDigit(ch))

  System.out.Prinln (ch+ " is a Digit ");
else if (Charactere.isWhitespace (ch))
 System.out.Println (" . . is a whitespace.");

else
System.out.Println (ch+" is a special character

}
Scanner.close();
}
}
```

(11) Differentiate between Static and nonstatic members including necessary examples. write a example Program that able to check either a number or string is pallindrome or not.

Difference between static and nonstatic members in java

Features	Static Members (static)	Non-static Members
Belongs to	Class itself (Shared among all objects)	Individual Objects (instances)
Memory Allocation	Allocated once per class	Allocated separately for each object
Access	Accessed using ClassName.memberName or within the same class	Accessed via an Object ObjectName.memberName
Usage	Used for constants, Utility methods, and Shared resources.	used when behavior / data needs to be differant for each instance.
Accessing Nonstatic from Static ?	Not allowed	Allowed

Example static Vs Nonstatic members

```
class Example {
static int staticVar = 10 ;
int nonstaticvar = 20 ;
 static void staticMethod ( ) {
 System.out.Println (" Static Method : ", + staticV
}      // non static not allowed

void nonStaticMethod ( ) {
System.out.Println (" Non Static Method :" +nonstati
System.out.Println (" Accessing staticVar inside
 non-static Method : " + staticVar ) ;
 }
}
Public class Teststatic {
Public static void main (String[ ] args) {
Example . staticMethod ( ) ;
Example obj = new Example ( ) ;
obj. nonstaticMethod ( ) ;   }
}
```

simplified Program to check if a String or number is a Palindrome.

```
Import java.util.Scanner ;

Public class Palindrome checker {
String
Static boolean isPalindrome (String str){
str = str.replaceAll ("\\s" , " ").toLowerCase () ;
return str.equals (new StringBuilder (str).reverse ().
                       toString ()) ;
}
Static boolean isPalindrome (int num) {
int original = num,  reverse =0 ;
mber
while (num>0) {
reverse = reverse* 10 + num % 10 ;
num/=10 ;
}
return original == reverse ;
}
```

simplified Program to check if a string or number is a Palindrome.

```
Import java.util.Scanner ;

Public class PalindromeChecker {
Static boolean isPalindrome (String str){
str = str.replaceAll ("\\s", " ").toLowerCase ();
return str.equals (new StringBuilder (str).reverse ().
                    toString ());
}
Static boolean isPalindrome (int num) {
 int original = num,  reverse =0 ;
 while (num>0) {
  reverse  = reverse* 10 + num %10 ;
  num/=10 ;
  }
 return original == reverse ;
}
Pubic ~~class~~static void main (String[] args)
Scanner sc = new Scanner (system.in);
// checking a string
System.out.print (" Enter a String : ");
System.out.println (is Palindrome (scanner.nextline())
    "Palindrome" : "Not Palindrome") ;

// checking a number
System.out.print (" Enter a number; ");
System.out.println ( is palindrome (scanner.nextI
"Pallindrome" : "Not Pallindrome" ) ;

Scanner.close() ;
}

}
```

(11) what is called class Abstraction and Encapsulation ? Describe with the example. what the the difference between abstract class and interference.

1. Abstraction

Abstraction is the fundamental object oriented Programming (oop) concept that focuses on hiding the implementation details while exposing only the essential functionalities. It helps reduce complexity and increase reusability. Abstraction canbe achieve using abstract class and interfaces.

2.Encapsulation

Encapsulation is the Process of binding data (variables) and Methods into a single unit (class) and restrictin direct access to some details using access modifiers (Private, Protected, Public)

```
// Abstract Class //
abstract class vehicle {
abstract void Start() ; // abstract method (no implem
}

// Concrete class implementing abstraction
class Car extends vehicle {
@ override
void start (){
system. out. Println ("Car is starting with key
                              ignition. ");
  }
  }

Public class Main {
Public static void main (String [] args) {
Vehicle mycar = new Car();
mycar. start ( ); // output: Car is starting
                        Key ignition.
}
}
```

```
// Encapsulation
Class BankAccount {
Private double balance ;   // Private variable

// construction

Public BankAccount (double initialBalance) {
  this. balance = initialBalance ;
 }

// Public method to access Private variable
Public double getBalance ( ) {
return balance ;
}
// Public method to Update Prive variable
Public void deposit (double amount) {
if (amount >0) {
balance += amount ;
  }
 }
}
Public class Main {
Public static void main (String [ ] args) {
BankAccount = new BankAccount (1000) ;
account. deposit (500) ;
System.out. Println (" Balance : " + account. getBalance()
}                                      // output balance = 1500
}
```

* What is the difference between abstruct class and interfercence.

Feature	Abstract class	Interface
Methods	can have both abstract and concrete (implemented) methods.	only abstract methods, can hav default/static methods
Fields	can have instance variables. (with any access modifier)	can only have Public, static, fin variables.
Constructor	Can have constructors	cannot have constructors
Mutiple Inheritance	Supports single inheritance	supports multiple Inheritance.
Access Modifiers	Methods canbe Public, Protected or default	All methods are Public by default
Use Case	Used when we need Partial abstraction	Used for full abstraction and defining a contra for multiple classe

(14) What is the significance of BigInteger ? write a Program which gives a method that can return the factorial of any Inteager.

In java, the BigInteger class is used for Performing arithmetic operations on very large inteagers that are beyond the range of Primitive data types like int and long. Since long in java can Store values up to $2^{63}-1$, computations involving large numbers (like factorials of big numbers) require BigInteager.

Code

```
import java.math.BigInteager ;
import java.util.Scanner ;
Public class FactorialBigInteager {
Static BigInteager factorial (int num){
BigInteager result = BigInteger.ONE ;
```

```
 for (int i = 2 ; i <= num ; i++) {
result = result.multiply (BigInteger.valueOf (i));
}
return result ;
}

Public static void main (String [] args) {
Scanner sc = new Scanner (System.in) ;
System.out.print ("Enter an integer : ");
int number = sc.nextInt ( ) ;

BigInteger fact = factorial (number) ;
System.out.println ("Factorial of " + number
" is :\n" + fact ) ;

sc.close() ;
}
}
```

(16) Polymorphism in java is the ability of an object to take on multiple forms. It allows a single interface to be used for different underlying forms (data types). Two primary types of polymorphism in java are :

1. Compile time Polymorphism (Method overloading) :- The method to be executed is determined at compile time.

2. Runtime polymorphism (Method overriding) :- The method call is determined at runtime .

Dynamic Method Dispatch : (also known as runtime polymorphism) is mechanism in java where method call to an overridden method is resolved at runtime rather than compile time. This allows a superclass reference varriable to refer to a subclass object and execute the overridden method of the actual subclass.

Example :-

```
// Parent class
Class Animal {
void makeSound () {
System.out.Println ("Animal makes a sound")
 }
}

// child class 1
Class Dog extends Animal {
@override
void makesound() {
System.out.Println ("Dog barks");
 }
}

// child class 2
Class Cat extens Animal {
@override
void makesound() {
System.out.Println ("Cats meows");
 }
}
```

```
Public class PolymorphismExample {
Public static void main (String[] args) {
 Animal myAnimal ; // superclass reference .
 myAnimal =new Dog() ; //  dog object
 myAnimal . makeSound() ; // output : dog barks

 myAnimal =new Cat() ;
 myAnimal . makeSound() ;
  }
 }
```

17) Difference between ArrayList and LinkedL

in Java :

Both ArrayList and LinkedList implement the List interface in Java, but they have different underlying data structures and Performance. characteristics.

Operation	ArrayList	LinkedList
Underlying structure	Dynamic array	Doubly linked list
Access (get/set)	O(1) (direct indexing)	O(n) (Sequenti traversal)
Inserting (add at end)	O(1)	O(1)
Insertion (add at middle /beginning)	O(n) (shifting elements)	O(1) (if Pointer Position is know) O(n) (traversal req
Deletion (remove from end)	O(1)	O(1)
Deletion (remove from middle / beginning)	O(n) (shifting elements)	O(1) (if Pointer i Known) O(n) (traversal required

Memory overhead	Lower	Higher

When to use ArrayList vs LinkedList.

Prefer ArrayList when :

- Frequent random access is required. O(1)
- Memory efficiency is a concern .
- Appending elements in the Primary operation O(1).
- Sorting is required often.

Prefer LinkedList when :-

- Frequent insertions and deletions at arbitary Positions .
- You are working with very large datasets.
- Memory Overhead Is not a Primary concern.

Performance Implications for Large Datasets :-

- **Access Performance** :- ArrayList scales better because direct indexing is fast, whereas Linkedlist required traversal.

- **Insertion and Deletion**:- Linked list excels when frequent insertions / deletions require but traversig the list to find the insertion Point can still be costly.

- **Memory Usage**:- LinkedList uses significantly more memory due to the additional node Pointers, which becomes Problematic for large databasets.