DATE

OOPS in Java by Kunal Kushwaha

classes, Objects, constructors & keywords.

- new → dynamically allocates memory and returns a reference to it for the variable.

- Student Student1 = new Student();
  - Student Student1 → compile time
  - new Student() → Run time

- Default value of String is null.
  - " " " int is 0.
  - " " " float is 0.0.

- constructor is a function which defines what needed to be done once you create the object of a class.

- Return type of the constructor is the class itself.

- When a method is declared static, meaning it belongs to the class and not to any specific instance of student.

DATE

- You can call a constructor from the another constructor using this

```
Class Student { String name;
    Student ()
    {
        this ("Mayesh")
    }
Student (String name)
    {
        this.name = name;
    }
}
```

- Primitive data types are not implemented as objects in java.

Stack Memory → • Stores method specific values
• local variables
method calls
• Stores primitive values and reference to objects.

Heap Memory → • Stores objects and class instances.
• Stores actual objects.

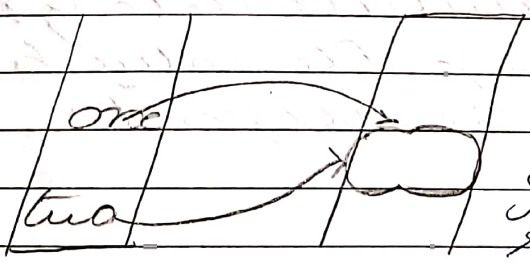
→ new allocates memory for an object during the runtime.

DATE

primitive data type	non-primitive data types
Basic data types that stores values directly	objects that store reference to memory
int, char, bool,	String, Array, Classes
Stored in stack memory	Stored in heap memory

→ non-primitive data types do not store the actual values directly, Instead they store a reference (memory address) that points to the actual values in a heap memory.

```
Student one = new Student();
Student two = one;
```



what you have done above is you made two reference variable to point the same object, So, the change in one will affect the other one also.

→ constructors can't have a explicit return type, if it's has a explicit return type then it's not a constructor and it's a method of the class

DATE

→ Java only supports call by value not call by reference.

→ When you pass a primitive and non-primitive data type to a function, and modify that variable in that function, the changes will stay only in that function and the original variable will not be affected.

Final

→ When a variable is declared as final it don't allow modifications/changes.

→ So, it is compulsory to always initialize the final variable while declaring.

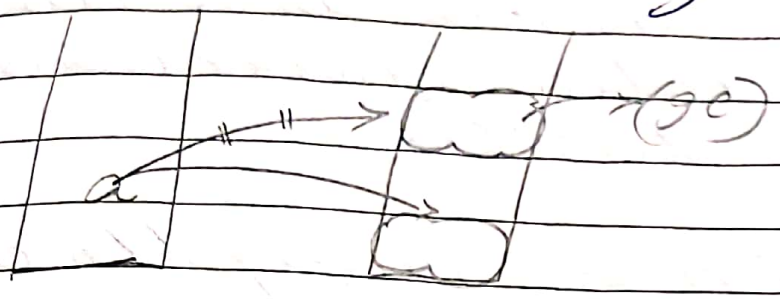
→ The above rules are only applicable to the primitive data type

→ But for non-primitive datatype you can change the value but you cannot reassign it.

```
Student Std2 = new Student();
final student Std1 = new Student()
   Std.name = "new name"
                ↳ modification are allowed
Std1 = Std2;
        ↳ Error, reassigning is not allowed.
```

DATE

→ When a non-primitive is final you can't reassign it.



→ When a reference variable changes it's reference point, the garbage collector in java will destroy that object, making the new object is only available in the loop.

Finalise.

→ When you want to show some message during the deletion of object using garbage collector you can call the destructor using finalise.

→ You can't tell when to delete in a memory, But you can tell what to do while deleting.

Refer the code //

DATE

Package

→ A package in java is a way to group related classes and interface together.

→ It helps in organizing the code and avoiding name conflicts.

→ Packages are stored in hierarchical mode. You can't create a two classes with same name under the same package

↳ pack1
demo.java
demo.java ⊗ error
↳ pack2
demo.java ✓

→ If you want to run a java package, first compile the pack

directory Structure

pack/
a.java
b.java

javac pack/*.java
↳ will compile the all java files inside the pack folder

java pack.a → To run the a.java file.

this: (keyword)

- Access instance variable (this.radius)
- Call another constructor in the same class
- Passed current object as a constructor: func(object)? sop(obj.radius) func(this)

Static

- Static variable belongs to the class, not to any specific objects
- All objects of the class share the same static variables.
- A change in the static variable affects all objects, since they refer to the same memory location.
- As they are not associated to any specific objects, they can be accessed using the class name itself.
- As they aren't dependent on the class, they can be accessed even the objects has been created.
- Method declared as static can be accessed without even creating the object for the class, Because they are independent of the objects.

DATE

→ You can't access non-static method (or) variable inside the static method because static method doesn't depend on the object where as non-static objects depends on the objects.

→ But non-static method can access the static method (or) variables.

→ You can access the non-static method (or) variable inside the static method by creating the object for the non static method and accessing / by entering the non static method using that object

→ you can't use "this" in the static methods because this refers to an object and whereas static method doesn't depend on a object

→ A static block in java is block of code inside a class that runs once when the class is loaded into memory. It is used to initialize static variables (or) run startup code

→ Static methods are class level method so it is always resolved during the compile time.

DATE

→ Static block will run only once declared it

→ when the inner class is dependent on the outside class. But when the inner class is declared as static, the inner class becomes independent of the outside class.

→ So, to access the inner class, you don't want to create object for the outer class

→ But static class, doesn't say that, it's methods and the variables are static.

Singleton class

→ until the constructor of the class is executed no objects (or) is created.

→ Singleton class allows you to create only one class object for their classes. no two objects of the same classes are allowed.

→ In static methods you can't use super() also because super needs an instance, while static method belong to the class and don't have access to this (or) super.

DATE

Inheritance

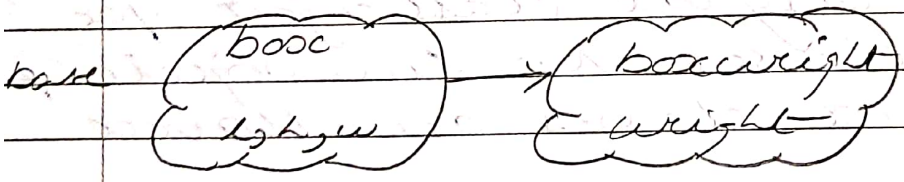
→ A concept in oops where one class (child) inherits properties and methods from another class (parent). Helps in code reusability and reducing duplication.

→ While using the child class, the parent class variables are also need to be initiated.

→ Even though it's a child class, it can't access the methods and variables declared as private.

Box box = new boxweight(2,3,4,5)

The reference variable type is Box
The object it referring to is boxweight.



The box can access the l,b,w but not the weight even though it's referring the boxweight object.

DATE

Because, the variable/method that can be accessible is determined by the type of reference variable and not by the object it referencing to.

* You can't reference variable of child class and try to reference the object of parent class

Boxweight box = new Box(); ⊗

It shows error. Through the Boxweight can all the attributes of both the Boxweight, Box class Because, there may be variables/methods that child class only have. And the parent class have no idea about these extra attributes and methods in the child class!

Super

→ used to access the constructor of the parent class from the child class.

→ Every class inherits from the object class of java.

→ It can also used to access the parent class variables from the child class.

DATE

→ If there is no argument constructor in the parent, we don't want to call the parent class constructor from the child class. explicitly because, whenever the child class constructor executes, it will automatically executes the default parent class constructor.

→ but if

```
class parent {  int age;
                parent (int x)
    {
        this.age = x;
    }
}

class child extends parent {
    child (int a)
    {
        Super (a); // 100% need to
                   be done
    }
}
```

→ If there is a argument constructor in the parent class, then 100% we need to call the parent class constructor while passing the int argument.

DATE

```
class
parent { int x;

    parent(parent obj) { obj.x; }
}
class
child extends parent {
    child(child obj)
    {
        super(child);
    }
}
```

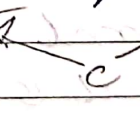
→ When you pass child object to the parent constructor, it is automatically treated as a parent type because a child class object is also a type of its parent class

→ Java automatically updates child to parent while calling super() which is known as upcasting.

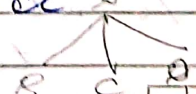
→ Downcasting is not allowed because the parent class object may not have all the properties of the child class.

Types of inheritance :-

- Single inheritance : a → b
- Multilevel inheritance : a → b → c
- Multiple inheritance : not in java

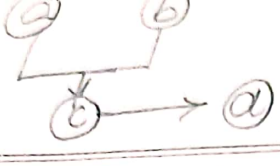


- Heranical Inheritance



classmate

PAGE



DATE

- Hybrid inheritance → Single + Multiple inheritance

→ A class can't be it's own parent / super class.

Final

Final variables must be initialized either:
→ At the time of declaration
→ In the constructor.

Types of final variables →

- Final instance variable (must be initialized once, either at declaration (or) in the constructor).

- Final Static variable (constant, shared access all instances).

- Final local variables (Once assigned, it's value cannot be changed).

→ Declaring class as final will implicitly declares all of it's method as final too. And prevents that class to be inherited.

DATE

polymorphism.

It is the ability of a method (or) object to take multiple forms. It allows the same method name to have different implementations either through method over-loading (compile time - polymorphism) method overriding (run-time polymorphism).

Types

→ method overloading (compile-time polymorphism) / (static polymorphism).

→ method overriding (run-time polymorphism).

→ operator overloading (a feature where the same operator behaves differently based on the operands)

Java doesn't support operator overloading except (+)

(+) for int addition

(+) for string concatenation.

Method overloading :-

→ Multiple constructors with different arguments (or) different order of the arguments.

→ During the compile-time it will decide, which method / constructor is used

DATE

to be run, hence this is known as compile time organisms.
polymorphism.

Method Overriding

→ Runtime / Dynamic polymorphism.

→ when a Superclass's method is overridden by the Subclass's method with some different definition.

Superclass	Subclass
area()	area()
SOP("Hi")	SOP("Hello")

Subclass obj = new Superclass()
Superclass obj = new Subclass()
obj.area() → //Hello
obj.area() → //Hello

→ obj has the ability to call both the parent class and child class area because the parameter / method that can access is determined by the type of reference variable and not the object it refers to.

now the method area() of Superclass is overridden by the area of Subclass()

(17) // objects type determine the version of the method that can be executed.

DATE

But, while considering obj, it's reference type is Superclass but it's able to call the Subclass's method because this known as method dispatch

" The methods that's get executed is determined at runtime, based on the actual object (child) and not the reference type (parent) "

But only overridden methods are resolved this way - variables are still accessed based on the reference type "

Methods are resolved at runtime (dynamic dispatch).

Superclass

Subclass

area()
{ sop("1,1") }

Superclass obj = new Subclass()
obj.area() // error

Because the area method is not overridden, hence the variable / method that can be accessed is determined by the type of reference variable and not by the type of object. Hence obj has no idea about the method area() which is in the Subclass().

classmate

PAGE

DATE

```
class anyclass {  String name;
                  void String toString()
                  {   this.name = str;
                      return name;
                  }
```

→ Every class will be extended from the object class, the object class will have the toString() method.

→ Now when you write the toString method, it is just overriding the objectclass toString() method which is known as method overridding done using dynamic dispatch.

	Compile-time	Runtime
Errors	Syntax & type errors	Exception (null pointer)
Resolution	Method overloading	Method overridding
Handled by	compiler (Javac)	JVM (Java virtual machine)
Ex:-	int x = "test"	int y = 1/0;
	Static Checking	Dynamic behaviour

DATE

Early Binding :- (Static Binding)

→ method call is resolved at compile time

→ Used in method overloading and static methods

→ Faster because method address is known before execution.

Late Binding :- (Dynamic Binding)

→ method call is resolved at runtime using dynamic method dispatch

→ used in method overriding.

→ Slower because JVM decides which method to call during the runtime.

Final

→ Methods declared as final can't be overridden, so it will always be early binding, prevents late binding / dynamic binding.

→ A class declared as final will not be inherited preventing the late binding.

PAGE

```
Superclass()          Subclass()
{                     {
  Static void()         Static void()
  { SOP "Hello"; }      { SOP "hi"; }
}                     }

Superclass obj = new Subclass();
obj.void()
  ↳ Hello
```

- We know that when a method is overridden, the method which is used to be executed is determined by the object type, in such way that, "hi" should be the output but here the output is hello, because, the,

"
In the context of method overridden, the method used to be executed is determined by the object type, but the method here called is Static hence, the static methods are independent of the object. So it calls the void method of the reference variable"

overridden depends on object, Static method doesn't depend on object. hence, static methods can't be overridden.

DATE

"you can inherit but you can't override" → when methods are declared as static.

@override → is an annotation in java used to indicate that a method is overriding a method from it's parent class.

Encapsulation

Encapsulation in java is the process of wrapping data (variables) and method functions into a single unit (class) and restricting direct access to the data.

Achieved using access modifiers.

Abstraction

Abstraction in java is the process of hiding implementation details and showing only the necessary feature to the user. It is achieved using Abstract classes and interface.

When a variable is declared as private it can be accessed using the getter and setter method.

PAGE

DATE

Encapsulation	Abstraction
→ protect data from direct access	Simplifies complex systems
→ using private variable and public getter/setter methods	using abstract classes and interfaces.

Data hiding → Restricting the access.

Encapsulation → Data hiding + getter/setter methods

→ Bundling the data and methods while controlling the access.

Access Modifiers

protected

→ when you declare a variable/method as a protected in a class.

→ they were accessable with in the class

→ they were accessable from different class

DATE

→ But when it comes to different packages, you should create the sub class of the main class / subclass of that parent class to access those protected variables / methods.

pack1

```
class parent
{
protected int
        i=10;
}
```

pack2

```
class child
extends
        parent
{
psvm(String
{
parent par = new
        parent();
child chi = new
        child();
error ① ← "par.i";
valid ② ← chi.i;
}
```

→ ② is valid because it extends the parent class, in different packages they can be accessed

→ ① Even though it's a parent class object, it's not able to access it in different package

→ Consider if ① gets executed then what is the difference b/w the public and protected

-d?

DATE

→ public → can access data variables with out subclass in different packages

But

→ protected → can access in different packages by only using the childclass.

packages

Types :

- user-defined packages.
- In-built packages

In built :-

- Java.lang → default package
  - no need to import
  - contains classes like String, Math, Integer
  - Automatically imported in every java program
- Java.io → Input output of a file
  - Streams, buffer, reader
- Java.util → Utility functions & datastructures
  - Arraylist, collections
- Java.awt → Java GUI, frame, Button
- Java.applet → small program embedded in web browser
- Java.net → networking & Internet communication
  - socket, URL etc...

DATE

Hashcode

- A method in java that returns a unique Integer for an object. It is used in hash based collections like HashMap, HashSet, HashTable for efficient search -ing and retrival.
- If two objects are equal (equals () returns true), they must have the same hashcode ().
- If a class overrides equals () it must also override hashcode () for consistency.

```
@override
public int hashcode ()
{
    return id;
}
```

→ So in the above code we are overridding the hashcode method, to return the value of id, instead of returning the random unique number.

instanceof (operator) checks whether an object is an instance of a specific class or it's subclass.

object instanceof classname → returns true/false.

DATE

== (equality)	.equals() (method)
→ Compares the memory addresses	Compares the content (values) in it
→ Can't be over riddes	Can be over riddes.
→ Applicable for primitive data types.	→ not applic-able for the primitive data -types.

```
class Student { int num;
@override
public boolean equals(Object
                      obj)
{ return this.num ==
        ((Student)obj).num;
}
```

In the above code we are over riding the equals method, where we accept any object and convert that into the students object and checking whether the conte-nt (num) of two objects are same (or) not.

private

private methods are not inherited making it impossible to over-ride them.

DATE

5th video on

OOPS

Abstract methods

→ If a class contains one (or) more abstract methods then 100% sure that it's class should also be abstract.

→ All the abstract methods of the parent class should be overridden by the child class.

→ For abstract classes you can't create objects, even if you create an object for the abstract class, you should override all the abstract of the class in the object creation.

```
{ parent mom = new parent(){
    @override
    void care(){ }
    @override
    void patience(){ }
  };
```

(1)

→ variables can't be declared as abstract.

→ It's not compulsory to assign the variables of abstract parent class from child class,

DATE

Anonymous class

- It is a class that has no name and it is declared inside another expression (usually inside a method)
- It is used when you need a one-time implementation of a code/class.

Q is an example of creation of anonymous class

→ It is create a subclass of a parent class without name

→ Overrides career() and partner() on the spot

→ The object is type of parent, but it belongs to the anonymous class.

- It is one-time usable.
- You cannot create another object of this anonymous class because it has no name.
- If you need to reuse the same functionality, you must create a named subclass instead.

→ You can create constructors for the abstract classes. But you can't create the abstract constructor (method) for the class.

DATE

→ Abstract methods can't be a static methods, because Abstract methods are used to overwritten whereas, static methods are not object dependent hence they can't be overridden.

→ Static methods can be created inside the abstract classes. but should always contains definition.

→ Abstract classes can also have normal methods and these methods can also be (optional) overridden in the extending childclasses.

```
abs parent {
abs void fun()
{
}

class child extends parent {
void fun()
{ sop("Hi"); }
}

parent obj = new child();
```

→ objects of abstract classes can't be created, but it can be used as the reference variables.

→ As you know, during the overridden methods, the method which is used to be accessed is determined by the type of object, that the reference variable is pointing to not by the

-> Abstract class variables are inherited by child class.

DATE

type of reference variable. And you know 100% a child class of abstract class (parent) will be overriding the methods of parent class. Hence overriding is done. So the (obj) reference variable can access the methods of childclass.

-> Abstract classes can't be final classes because the abstract classes are used to be inherited, whereas the final class can't be inherited. Hence a class can't be both abstract and final.

-> You can have both static variables in the abstract classes. & final

100% like the abstract but Interface.

-> Interface are implemented using implements which is used to allow/achieve the multiple inheritance in Java.

-> We cannot create objects of an instance directly because interfaces doesn't have a complete implementations (they only define the method signatures).

-> Use a class that implements the interface to create the object.

classmate

PAGE

DATE

Abstract	Interface
° A class only inherits a single super class	A class can inherit/implement more than one interface,
° All the variables can be static (or) non static, final (or) non final	° All the variables are final by default and also static
° Abstract methods can't be private (private method can't be accessed (or) overridden in a subclass).	° All the methods in interface are public by default.

→ All the methods in an interface are public by defaults.

→ Also when implementing the interface in a class, the method must remain public

→ you can't make it protected (or) private.

→ An interface can't implement another interface, it can only extend another interface.

DATE

(*) Advantage of Interface:

• Multiple unrelated classes can implement an interface, the methods must be accessible from everywhere.

→ Interface doesn't care about which parent class you belong to, (or) which childClass you're, it sees whether you implement that interface or) not. if you're implemented from that interface they can override the methods in the interface and can use that methods. whether you're in different package, different class, different subclass, you can override the methods of the interface.

→ So, if a method in interface is declared as private, then any implementing class can't override it breaking the use interface, so methods in the interface are not allowed to be private.

→ The methods in interfaces are public by default, even you specify public (or) not.

void display() { equal
public void display() }

classmate

PAGE

DATE

→ In interfaces, only default, static, private methods are allowed to have functions definition.

```
void display()
{
SOP("Hi");
}
```
↓ error

By default all methods are public"

```
default
void display()
{
SOP("Hi");
}
```
↓ works

"default methods can have fun. definition"

→ A class cannot extend an interface, it can only implement an interface. However, an interface can extend another interface".

```
interface A
{
fun();
}
```

```
interface B extends A
{
dis();
}
```

```
class C implements B
{
@override fun() { SOP("Hi"); }
@override dis() { SOP("Hello"); }
}
```

→ interface A is extended by the interface B, so while the class C implements B, it should override the both the functions of class A and class B.

DATE

→ Static methods in a interface should always have the function declarations and definitions.

→ methods declared as defaults in interface can have the function definitions.

```
class A                    class B extends
{                          A {
static void                class main {
fun(){                     B obj = new B();
SOP("Hi");                 obj.fun();
}                          }
}                          }
```

- Now the class A methods are inherited to the class B and thus class B objects can access the fun method.

```
interface A {              interface B extends
                           A {
}                          }

                           class main {
                           B obj = new B();
                           obj.fun(); → (1)
                           A obj1 = new A();
                           obj1.fun(); → (2)
                           A.fun() → (3)
```

① → Like as inheritance, the abstract methods are not inherited to the subclass interface. But default methods are inherited. The method fun in the interface A, will not be inherited to the interface B. Hence, interface B can't access the fun method.

② → objects for the interfaces can't be created.

③ → Hence, static methods are directly accessed/used using the interface itself.

```
interface A
{
default fun()
{
SOP("Hi");
}
}
```

```
class B implements A
{
PSVM(String arg)
{
A obj = new A();
obj.fun(); → ①

B obj = new B();
obj.fun(); → ②
}
}
```

① we can't create objects for an interface, so we can't access the fun() method.

DATE

/interface

② default methods belongs to instances of implementing class and they can be used/called by the instances of the implementing class. These default methods can also be overridden in implementing class. If the method fun() is not default then the implementing class should write the function definition for that method while implementing.

default static void fun() { SOP("Hi"); }

→ error,
In the context of interface,

- static method belongs to the interface itself, meaning they can't be inherited (or) overridden.

- default methods belongs to instances of a class that implements the interface, meaning they can be overridden.

→ Since, one is inherited (default) and the other is not (static). Java prints them from being combined

→ default keyword is only used with interface methods and not in normal class methods.

DATE

Polymorphism.

Method overloading

→ Java looks for a method with the exact parameter type cast match.

→ If no exact match is found, Java will automatically upcast (also called widening conversion) (int is upcasted to float)

→

If widening isn't possible, Java tries unboxing (eg:- int to Integer)

→ Instance variable are implicitly accessed using the current object (this); so explicitly using "this", only necessary when there is a naming conflict ("Eg:- when a method parameter has the same name as the instance variable).

↳ X Ryan overloading - Bt

(*) Important properties of Interface.

→ Interface will have only abstract methods

→ variables declared in interface are final by default

DATE

→ Interface can have only static and final variables

→ Interface can't provide the implementation to the abstract class.

→ An interface can extends only another interface.

→ Members of a interface are public by default.

→ You can't modify an interface variables because they're public, static, final by default

- public (can be accessed from anywhere)
- Static (not specific to any object)
- Final (constant)
- Hence, interface are stateless

→ Interface allow "unrelated classes" to share a common behaviour.

→ Static members (both members and variables) belong to the interface itself, not to any instance / subclass

DATE

```
interface parent
{
  static i=10
  static void
    fun(){
      sop("Hi");
    }
}
```

```
interface child
extends parent
{
}
```

```
class main
  psvm(String[]args)
  {
    SOP(child.a); //error
    SOP(child.fun); //error
    SOP(parent.a);
    SOP(parent.fun);
  }
```

→ When a interface extends another interface, it does not inherit the static members from the parent interface

→ The child interface can still access the static variable using the parent interface name, but it does not get a copy of it.

You can use the interface type variable to invoke the method of the different DATE
implementing class

```
interface Animal {
    void makeSound();
}

Dog
class implements Animal {
    public void makeSound()
    { SOP("Woof"); }
}

class Cat implements Animal {
    public void makeSound()
    { SOP("Meow"); }
}

public class Main {
    psvm(String[] args)
    {                                   Run time
        // dynamic lookup of method at
        Dog a = new Dog();
        Cat b = new Cat();
        a.makeSound();
        b.makeSound();
```

Animal ref;
ref = new Dog();
ref.makeSound();
ref = new Cat();
ref.makeSound();

Instead of this we can do this.

- This process is similar to using a superclass reference to access a subclass object.
- Any instances of a class that implements the declared interface can be referred by such a variable.

DATE

→ The method to be executed is looked by dynamically.

→ The nested Interface can be public private (or) protected

```
class A
{
    public interface nested IF
    {
        void fun();
    }
}
// example of nested interface.
```

⇒ If two interfaces have the same default method, the class must override it.

DATE

oops - 6th video.

Generics allows you to create, classes, interface and methods that can work with different data types while ensuring the type safety at compile time.

→ type

class c <T> parameter

<T> is known as type parameter.

public class mobile devices <T> {
    private static T os; // error
}

class Main { psvm (String args[]) {
    Mobile devices <String> phone;
    Mobile devices <Int> Oppo;
}

→ you can't declare static fields whose type are type parameters.

→ because, as we know in the above the os's types will get changed based on the value assigned to T.

→ And thus T can be decided on the object, Each object will have different T.

-> But when we are trying to use static it doesn't depend on the objects, hence it will apply the functionalities <T> type parameter.

wildcard

class Array <T extends Number>
{
. . . }

-> A wildcard in java generics represent an unknown type. It is used when you don't know (or) don't want to specify the exact type while allowing flexibility in generic parameters

-> In the above the <T> can accept type of number (parent class of integer, Float, double) and also it's subclasses (Integer, Double, Float) etc...

You can't create an object of T in generics?

Because

class Summa <T>
{
public void creInstance()
{
T obj = new T(); // creat
}
}

DATE

- <T> is just a placeholder at compile time
- when the program runs, java knows I and doesn't know what type to create.
- Since Java doesn't know how much memory is needed it disallows new T().

→ Interface can also be generic.

comparable → compareTo

(%) Refer the comparable-example!

Exception handling.

Exception	Error
An issue that can be handled at runtime	Serious issues that are not meant to be handled
Can be caught and modified using try-catch	Cannot be recovered, causes the program to be terminated
nullpointerexception is exception	out of memory error, StackOverflow