PROPERTIES OF INTERFACE

* KEYWORD → Interface
* all methods are implicitly public & abstract.
* It's method can't be static
* It's methods can't be final, since abstract.
* A interface extends one or more interface.
* A interface CANNOT implement another interface or class — OBVIOUS BUD...
* Interface types can be used POLYMORPHICALLY

Variable declared;
static final <type> <name> = <val>;
method: <retype> <methodname> (Parameter list);

class <name> implements <interface> {
}

<Interface name> <name>;

INTERFACES - II
#LECTURE - 21

There are inbuilt interfaces like

ITERATOR ⇒ to run through the collection of objects without knowing how the objects are stored. Eg: Array, list, bag or set.

CLONEABLE ⇒ To make a copy of existing obj via clone() method on the class object.

COMPARABLE ⇒ To make a total order of objects.

ITERATOR

java.util package.

hasNext()
next()
remove()

CLONEABLE

A class x that implements the cloneable interface tells the obj of the class x can be cloned.

* The interface is empty, it has no methods.
* It returns a identical copy of an object.
  - ⇒ SHALLOW COPY → default
  - ⇒ DEEP COPY → preferably
* Prevention of cloning,

  If can't clone it would throw an exception.

  "cloneNotSupportedException"

SERIALIZABLE

* A class x that implements the serializable interface tells client that x object can be stored on a file or other persistent method.
* The interface is empty, it has NO METHODS

COMPARABLE

* helps in comparison.

POINTS TO NOTE

Declaration of interface ~ class.

Interface definition ⇒ interface declaration + body

name, whether it extends from another interface.

Interface SHOULD BE PUBLIC.

↳ IF NOT it will be visible to the classes defined in the SAME PACKAGE as the interface.

MEMBERS, final, abstract.

class classname [extends superclass] implements intf1, intf2....
{ }

PUBLIC.

if 2 or more interface ⇒ seperated by comma.

if a class implements 2/more interface that DECLARE the SAME METHOD, then that method will be used by clients of either INTERFACE.

* A method that implements an interface must be declared public.

* class that implement interface can have its own METHOD too.... obvious....

PARTIAL IMPLEMENTATION

if a class includes an interface but NOT FULLY implement/override the METHODS required by that interface.... THEN, that class must be declared ABSTRACT.

show()
callback()

```
abstract class In implements callback {
    int a,b;
    Void show() { }
}
```

Here, the class In does NOT imp callback().
∴ declared abstract.

* class that inherits in must implement callback() or declared ABSTRACT ITSELF.

NESTED INTERFACE

↳ an interface can be declared as a member of a class or another interface. Such interface is called NESTED INTERFACE.

→ Nested interface should be declared public.
→ Nested interface should specify its class name to avoid ambiguity. ⇒ classname.interface name

Also during writing implements... we do thus!

```
class A { }
class B implements A.NestedInterface {

}
class Demo {
    A.NestedInterface d = new B();
```

WHY INTERFACE?

→ to share common data
→ to inherited in multiple sense.
MULTIPLE INHERITANCE
* Runtime polymorphism.

You can use interfaces to import shared constants into multiple classes by simply declaring an interface that contains variables that are initialized to desired value.

RANDOM ⇒ Random r1 = new Random();
nextdouble ⇒ RANGE: 0.0→1.0

INTERFACE CAN BE EXTENDED

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

* when a class implements an interface that inherits another interface, it must provide implementation for all methods required by interface inherited chain.

MULTIPLE INHERITANCE ISSUES:

* Java does NOT support multiple Inheritance
* Class stores state info in instance variable.
  BUT Interface can't.

```
Interface Alpha {
    reset()
}

Interface Beta {
    reset()
}

class A Implement Alpha, beta {
    // which reset is used
    // what if class A has it's own reset.
}
```

"CONFLICT RULES" - Java gave us.

RULES

* class implementation overrides interface default im method.
* error occurs if both have same method.

interface α { }
Interface β extends α { }
// β's method is used

* α's can be accessed by super()

↳ Parent Int Name. super . methodName();
Eg: Alpha . super . fun();

DEMONSTRATION - 9
#LECTURE 22

@Interface ⇒ implicitly members are declared
mem → final, public & static → Can be accessed without obj. creation
method → public & abstract.

Eg ② ⇒
C c = new C(); ⇒ Error
C c[]; // Declaration is possible
C c[] = new C[3]; // array of obj is PERMITTED!

* Declaration is NOT allowed in interface.
* Private access specifier is NOT allowed. ↑ Inside.

* interface a implement C → abstract class INSTEAD of a interface/class ... ERROR.

* DON'TO declare method final & NOT STATIC TOO!!
↓
Then NOT accessable, won't be able to override that method.

When a extend 2 interfaces that is one implement inherits from other. Then that class should override all the methods of both

```
interface I1 {①}
interface I2 extends I1 {②}
class A implements I2 {
        ①
        ② // override
}
```

We can have multiple inheritance by $I_1, I_2 \ldots$ Extending one class & implating other int.

class B extends A implements I_1, I_2

```
Intname name;
name = classThatImplement it;
```

↳ Then use this template to implement methods of that implented class.

Abstract class are extended by sub class or Inherited. Then used.

EXCEPTION HANDLING-1
#LECTURE-23

ERRORS IN A PROGRAM

* Compile time error → Syntax, So on....
* Run time error
  → checked
  → maintained
  → manipulated

} Exception handling comes here.

REASON:
→ Invalid data
→ Divide by ZERO
→ out of bound.
→ Illegal change of thread.
→ Null obj reference.
→ A file that can't be found.
⋮
etc.

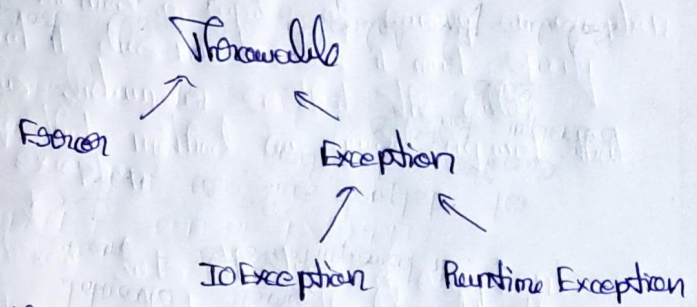
Errors ≡ Exception
↳ checked by JAVA RUN-TIME ENVIRONMENT
↓
It throws "an object corresponding to that"

m(int x, int y)
m(4.5, 3.5)

Arithmetic Exception.

Illegal Argument Type

Java offers classes hierarchy.
These are defined java.lang package



all these are classes & subclasses.

Java provides ⇒ "Run time Error Management"

KEYWORDS

try { }
catch { }
throw
throws
finally { }

Exception handling is done by this 5 keywords.

During the execution of the program, when exception condition occurs

↳ An obj of the respective class is created & thrown in with which method that caused the exception."

That method may choose to catch the exception & then "can guard against premature exit" or may have a "block of code to execute"

```
method {
  try {
      Statment(s) that may cause exception(s)
      throw exception obj
  }
  catch {
      ....
      Statmen handles exception
  }
  finally {
      Statment(s) that executes an exit
      of the exception handling
  }
}
```

① try with a single catch
② try with multiple catch
③ Multiple exceptions with one catch.
④ try-catch with finally
⑤ throw in try block
⑥ try within try

SIMPLE TRY-CATCH

```
try {
}
catch {
}
```

try block.
catch block.

if error occurs, we know that till that part code runs and then breaks.... we handle it and let it continue.

EXCEPTION HANDLING-2
#LECTURE-24

TRY WITH MULTIPLE CATCH

```
try {
    throw
}
catch {
}
```

null pointer exception
ArrayIndexOutOfBoundsException.

MULTIPLE ERROR WITH SINGLE CATCH

```
try {
}
Catch (Exception e)
    S.o.p (e.getmessage());
```

FINALLY IN TRY-CATCH

```
try {
}
catch{
}
finally {
}
```

If there is any error or NOT, then the finally block is "ALWAYS" gets executed.

e.tostring.

THROW IN TRY-CATCH: MECHANISH

" If a method is capable of causing an exception that it does NOT handle, it must specify this behaviour so that the caller of that method can guard against that exception".

↓

"throw" lists the types of exception that a method might throw.

↓

all the methods other exceptions that a method can throw must be declared in the throw clause.

↑

IF NOT, Compile time error occurs.

throws

type method-name (parameter-list) throws Exceptions-list

{

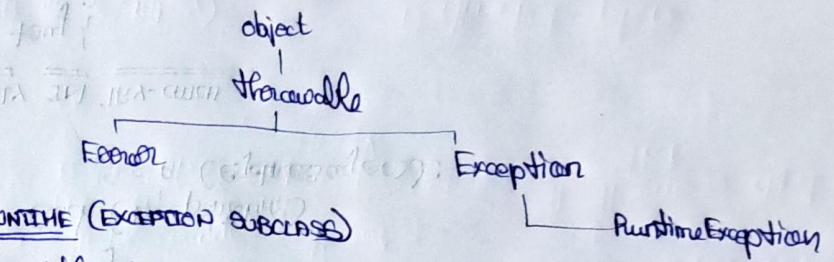
}

Exceptions-list: Comma seperated list of exceptions that a method can throw.

EXCEPTION HANDLING - III
#LECTURE - 25

There can be nested try-catch block exists

BUGS IN EXCEPTION HANDLING



RUNTIME (EXCEPTION SUBCLASS)

- → arithmatic
- → array index out of Bound
- → array store
- → class caste
- → illegal argument
- → security
- → index out of Bound
- → Negative array size
- → Null pointer
- → Number format

- → Incompatible class change.
- → String Index out of bound.

EXCEPTION

- → classNotfound
- → dataformat
- → illegalaccess
- → instantiation
- → interrupted
- → nosuchmethod
- → runtime

EXCEPTION METHOD

String getMessage()

Throwable getCause() → Cause of an exception represented as Throwable obj.

String toString() → returns name of class concat with getMessage()

Void printStackTrace() → toString() with stackTrace to System.err.

The error output stream.

USER DEFINED EXCEPTION

→ all exception must be a child of Throwable.

```
class myException extends Exception{
}
```

→ If you want to write a Runtime exception, you need to extend RuntimeException.

→ Similarly for exception, extend exception.

class → DataInputStream.