Employee e;

1. Salaried
2. Hourly
3. Commissioned
4. BasePlusCommissioned

```
e = new Salaried();
e. accept();
e.calculateSalary();

if(e instanceof BasePlusCommissioned)
{
BasePlusCommissioned emp = (BasePlusCommissioned)e;
emp.SalaryAfterIncrement();
}
```

SalaryAfterIncrement();

Revision

Abstract
1. Method
2. Class

```
abstract class Shape{
double area;

Shape(){

}

abstract void calculateArea();

double getArea(){

}

}
```

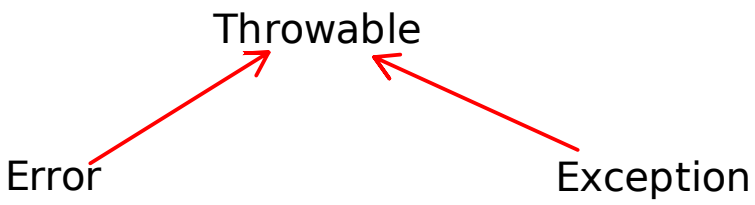
1. Fragile Base class Problem
2. It provides a set of protocols
3. Used to group unrelated classes together

```
interface  Acceptable{
public static final int num1;

public abstract void acceptData();

}
```

Exception Handling
1. Error
2. Exception



try
catch
throw
throws
finally

- In java, exceptions are divided into 2 categories

1. Error
    - It is caused by the runtime environment
    - It is recommended not to handle the errors.
    - OutOfMemoryError, StackOverFlowError,...
2. Exception
    - It is caused by the program in execution due to some wrong inputs or some errors caused in B.L.
    - It is recommended to handle the exceptions
    - we can handle the exception in 3 ways
    1. try with catch
    2. try with finally
    3. try with resource

1. try with catch-
    - Every try block should have atleast 1 catch block.
    - try block looks for the exception if generated by the statements kept inside it.
    - if exception is generated it will execute the matching catch block.
    - a single try block can have multiple catch block

2. try with finally
    - Every try block should have atleast 1 catch block, however if catch block is not provided then finally block can be added to such try block
    - finally block executes every time irrespective of the exceptions
    - It is used to release the resourses allocated.
    - If catch blocks are given then finally block should be the last block in the catch block series

3. try with resource
    - If catch and finally blocks are not provided then we can write the try block with resource
    - It is used for the resources which have implemented AutoCloseable interface
    - this block will close the resources when the try block finishes the execution

- Generic Catch Block
    - If we want to handle all the exceptions in the single catch block then
    use generic catch block.
    - the generic catch block handles the exception if type class `Exception` which is the super class
    of all the exceptions in java
    - generic catch block should be the last catch block in the catch block series

Checked Exception
- Exception class and all the subclasses except RuntimeException are all checked exception
- checked exceptions are mandatory to handle

UnChecked Exception
- RuntimeExecption classs and all its subclasses are unchecked exceptions
- unchecked exceptions are optional to handle

throw
    - to generate new exception
    - It can be used to generate checked as well as unchecked exceptions
    - to generate checked execption throw object of Exception class or its sub classes
    except RuntimeException class.
    - to generate unchecked exception throw object of RuntimeException class or its sub classes

throws
    - to route the exception from current method to the calling method
    - when we generate checked exception then it is mandatory to handle the exception
    in the same method in which it is generated.
    - to route such exceptions towrds the caller method use `throws` declaration

Custom Exceptions
1. Checked
2. unchecked

s