**Java 8 programs**

**Iteration of map:**

|  |
| --- |
| **// 1. Iterate map object using forEach**  Map<String, Integer>items = **new** HashMap<>();  items.put("A", 10);  items.put("B", 20);  items.put("C", 30);  items.put("D", 40);  items.put("E", 50);  items.put("F", 60);    //items.forEach((k,v) ->System.out.println(k +" : "+ v));  items.forEach((k,v) -> {  **if**(v> 40) {  System.***out***.println(k +" : "+ v);  }  });  **o/p:**  E : 50  F : 60  **// 2. Stream of skip and forEach**  IntStream.*range*(1, 10)  .skip(5)  .forEach(System.***out***::println);  **o/p:**  6  7  8  9  **// 3. Stream of sum of numbers**  **int**sum = IntStream.*range*(1, 10)  .sum();  System.***out***.println(sum);  **o/p:**  45  **// 4. Stream of sortd and findFirst**  Stream.*of*("ab", "nag", "scala", "java","mongodb")  .sorted()  .findFirst()  .ifPresent(System.***out***::println);  **// 5. Stream fromarray, filter,sorted and print**  String[] str = {"ab", "nag", "scala", "java","mongodb", "spring"};  Arrays.*stream*(str)  .filter( s ->s.startsWith("s"))  .sorted()  .forEach(System.***out***::println);  **// 6. Stream of mapaverage and ifPresent**  Arrays.*stream*(**newint**[] {2,4,6,8,10})  .map((e) ->e \* e)  .average()  .ifPresent(System.***out***::println);    **int**sum = Arrays.*stream*(**newint**[] {2,4,6,8,10})  .map(e ->e\*e)  .sum();  System.***out***.println("sum: "+ sum);  **// 7. Stream of sorted,filter and foreach**  List<String>list = Arrays.*asList*("ab", "nag", "scala", "java","mongodb", "spring");  list.stream().map(String::toUpperCase)  .sorted()  .filter(e ->e.length() > 3)  .forEach(System.***out***::println);  **// 8. Stream rowsfrom files, sorted, filter and print**  **Create demo.txt file**  nagendra  scala good programing language  java best programing  rabbitmq  mongodb  swagger  spring  cloud  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  filedata.sorted()  .filter(e ->e.length() > 15)  .forEach(System.***out***::println);  filedata.close();  **o/p:**  java best programing  scala good programing language  **// 9.Stream rows from files, filter and save into list**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  List<String>list = filedata.sorted()  .filter( s ->s.contains("spring"))  .collect(Collectors.*toList*());  list.forEach(System.***out***::println);  filedata.close();  o/p  spring  **// 10. Stream rows from files map,split,filter,count**  A,1,2,3  B,4,5,6  C,7,8,9  D,10,15  E  F,16,17,18,19,20  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  **long**count = filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .count();  System.***out***.println(count);  o/p:  3  **// 11. Stream rows from files map,split,filter,print**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .filter(e ->Integer.*parseInt*(e[1]) > 6)  .forEach(x ->System.***out***.println(x[0] +" " + x[1] +" "+ x[2] + " "+ x[3]));  filedata.close();  o/p:  C 7 8 9  **// 12. Stream rows from files map,split,filters, save into map**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  Map<String, Integer>map = **new** HashMap<>();  map = filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .filter(e ->Integer.*parseInt*(e[1]) > 2)  .collect(Collectors.*toMap*( k ->k[0], v ->Integer.*parseInt*(v[1])));  filedata.close();  **for**(Map.Entry<String, Integer>m : map.entrySet()) {  System.***out***.println(m.getKey() +" : "+ m.getValue());  }  o/p:  B : 4  C : 7  **// 13. Stream of reduce**  **double**result = Stream.*of*(1.0,2.0,3.0,4.0,5.0).reduce(0.0, (Double a, Double b) ->a + b);  System.***out***.println(result);  o/p:  15.0  **// 13. Stream of summarystatistics**  IntSummaryStatisticssummary = IntStream.*of*(1,2,3,4,5)  .summaryStatistics();  System.***out***.println(summary);  o/p  IntSummaryStatistics{count=5, sum=15, min=1, average=3.000000, max=5}  **// 14 iterate elements from stream and applyoptional class**  List<String>list = Arrays.*asList*("ab", "db", "c");  Optional<String>opt = list.stream()  .filter(e ->e.startsWith("ab"))  .findAny();  **if** (opt.isPresent()) {  System.***out***.println(opt.get()); // if value present get the value from optional  } **else** {  System.***out***.println("no value found");  }  opt.ifPresent(e ->System.***out***.println(e)); // if value is not found, ifPresent statement executed  List<String>list2 = Arrays.*asList*("ab", "db", "c");  String str = list2.stream().  filter(e ->e.startsWith("dd"))  .findAny()  .orElse("no value found");    System.***out***.println(str);  Optional<String>nonEmptyGender = Optional.of("male");  Optional<String>emptyGender = Optional.empty();  **// 15 Stream from list of objects**  **publicclass** Student {  **private** String name;  **private** Set<String>book;  **publicvoid**addBook(String book) {  **if** (**this**.book == **null**) {  **this**.book = **new** HashSet<>();  }  **this**.book.add(book);  }  **public** String getName() {  **return**name;  }  **publicvoid**setName(String name) {  **this**.name = name;  }  **public** Set<String>getBook() {  **return**book;  }  **publicvoid**setBook(Set<String>book) {  **this**.book = book;  }  }  Studentobj1 = **new**Student();  obj1.setName("mkyong");  obj1.addBook("Java 8 in Action");  obj1.addBook("Spring Boot in Action");  obj1.addBook("Effective Java (2nd Edition)");  Studentobj2 = **new**Student();  obj2.setName("zilap");  obj2.addBook("Learning Python, 5th Edition");  obj2.addBook("Effective Java (2nd Edition)");  List<Student>list = **new**ArrayList<>();  list.add(obj1);  list.add(obj2);  /\* List<String>listresult = list.stream().map( e ->e.getBook())  .flatMap(e ->e.stream())  .filter(e ->e.startsWith("Spring"))  .collect(Collectors.toList());\*/  Optional<String>opnResults= list.stream().map( e ->e.getBook())  .flatMap(e ->e.stream())  .filter(e ->e.startsWith("Spring"))  .findAny();  opnResults.ifPresent(System.***out***::println);  **// 16 Stream from list of objects**  **publicstatic** List<Student>getStudents(){  List<Student>list = **new**ArrayList<>();  list.add(**new** Student(11, 28, "Lucy"));  list.add(**new** Student(28, 27, "Tim"));  list.add(**new** Student(32, 30, "Daniel"));  list.add(**new** Student(49, 27, "Steve"));  **return**list;  }    Predicate<Student> p1 = s ->s.stuName.startsWith("A");  Predicate<Student> p2 = s ->s.stuAge< 40;  Predicate<Student> p3 = s ->s.stuAge< 40 &&s.stuName.startsWith("P");  List<Student>list = Student.getStudents();  /\* allMatch() method returns true if all the elements of stream satisfy the  \* given predicate, else it returns false  \*/  /\* This will return false because all student names do not start with "A"  \*/  **boolean** b1 = list.stream().allMatch(p1);  System.out.println("list.stream().allMatch(p1): "+b1);  /\* This will return true because all students have age less than 40  \*/  **boolean** b2 = list.stream().allMatch(p2);  System.out.println("list.stream().allMatch(p2): "+b2);  /\* This will return false because all the students do not satisfy the predicate:  \* Age must be less than 40 and name starts with letter "P"  \*/  **boolean** b3 = list.stream().allMatch(p3);  System.out.println("list.stream().allMatch(p3): "+b3);  //17 Stream Collectors groupingBy and counting Example  List<String> names =  Arrays.asList("Jon", "Ajeet", "Steve",  "Ajeet", "Jon", "Ajeet");  Map<String, Long> map =  names.stream().collect(  Collectors.groupingBy(  Function.identity(), Collectors.counting()  )  );  System.out.println(map); |
|  |

Multiple Sorting:

|  |
| --- |
| **public** **class** SortByMultipleField {  **public** **static** **void** main(String[] args) {  System.***out***.println("===== SORTING BY MULTIPLE ATTRIBUTES =====");  List<Employee> listEmployees = **new** ArrayList<Employee>();  listEmployees.add(**new** Employee("Tom", "Developer", 45, 80000));  listEmployees.add(**new** Employee("Sam", "Designer", 30, 75000));  listEmployees.add(**new** Employee("Bob", "Designer", 45, 134000));  listEmployees.add(**new** Employee("Peter", "Programmer", 25, 60000));  listEmployees.add(**new** Employee("Tim", "Designer", 45, 130000));  listEmployees.add(**new** Employee("Craig", "Programmer", 30, 52000));  listEmployees.add(**new** Employee("Anne", "Programmer", 25, 51000));  listEmployees.add(**new** Employee("Alex", "Designer", 30, 120000));  listEmployees.add(**new** Employee("Bill", "Programmer", 22, 30000));  listEmployees.add(**new** Employee("Samuel", "Developer", 28, 80000));  listEmployees.add(**new** Employee("John", "Developer", 35, 67000));  listEmployees.add(**new** Employee("Patrick", "Developer", 35, 140000));  listEmployees.add(**new** Employee("Alice", "Programmer", 35, 80000));  listEmployees.add(**new** Employee("David", "Developer", 35, 99000));  listEmployees.add(**new** Employee("Jane", "Designer", 30, 82000));  System.***out***.println("\*\*\* Before sorting:");    **for** (Employee emp : listEmployees) {  System.***out***.println(emp);  }    /\*Collections.sort(listEmployees, new EmployeeChainedComparator(new EmployeeAgeComparator(),  new EmployeeSalaryComparator(),  new EmployeeJobTitleComparator()));\*/    Comparator<Employee> employeeAgeComparator = (e1,e2) -> e1.getAge() - e2.getAge();  Comparator<Employee> employeeSalaryComparator = (e1,e2) -> e1.getSalary() - e2.getSalary();  Comparator<Employee> employeeJobTitleComparator = (e1,e2) -> e1.getJobTitle().compareTo(e2.getJobTitle());    /\* Stream<Employee> empStrems = listEmployees.stream().  sorted(new EmployeeAgeComparator()  .thenComparing(new EmployeeSalaryComparator())  .thenComparing(new EmployeeJobTitleComparator()));\*/    Stream<Employee> empStrems = listEmployees.stream().  sorted(employeeAgeComparator  .thenComparing(employeeSalaryComparator)  .thenComparing(employeeJobTitleComparator));    System.***out***.println("\n\*\*\* After sorting:");  empStrems.forEach(System.***out***::println);    /\*for (Employee emp : listEmployees) {  System.out.println(emp);  }\*/  }  }  **Employee Comparator Chains:**  **public** **class** EmployeeChainedComparator **implements** Comparator<Employee> {  List<Comparator<Employee>> list;    **public** EmployeeChainedComparator(Comparator<Employee>... comparators) {  **this**.list = Arrays.*asList*(comparators);  }  @Override  **public** **int** compare(Employee e1, Employee e2) {  **for**(Comparator<Employee> comparator : list) {  **int** result = comparator.compare(e1, e2);  **if**(result != 0) {  **return** result;  }  }  **return** 0;  }  } |

**Programs**

**Program: Divide the letter and numbers in string**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  String digit="";  String letter="";  String str= "nagendra123";  **for**(**int** i =0; i<str.length();i++) {  **char** ch = str.charAt(i);  **if**(Character.*isDigit*(ch)) {  digit+=ch;  }**else** {  letter+=ch;  }  }  System.***out***.println(digit);  System.***out***.println(letter);  }  }  **o/p:**  123  nagendra |

**Program: count the space and character**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  String str = "java is programing language";  **int** count = 0;  **for**(**int** i=0; i<str.length(); i++) {  **char** ch = str.charAt(i);  **if**(ch != ' ') count++;  }  System.***out***.println("no of characters: "+ count);  System.***out***.println("no of space: "+ (str.length() - count));  }  }  **o/p:**  no of characters: 24  no of space: 3 |

**Program: reverse method**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {    String str ="java";  String revStr = "";  **for**(**int** i=str.length()-1; i >= 0; i--) {  revStr+=str.charAt(i);  }  System.***out***.println(revStr);  }  } |

**Program: producer consumer design patteren**

|  |
| --- |
| **public** **class** ProducerConsumerDesignPattern {  **public** **static** **void** main(String args[]) {    BlockingQueue<String> queue = **new** LinkedBlockingDeque<>();  Thread producer = **new** Thread(**new** Producer(queue));  Thread consumer = **new** Thread(**new** Consumer(queue));    producer.start();  consumer.start();  }  }  **class** Producer **implements** Runnable {  **private** **final** BlockingQueue<String> blockingQueue;    **public** Producer(BlockingQueue<String> queue) {  **this**.blockingQueue = queue;  }  @Override  **public** **void** run() {  **try** {  **for** (**int** i = 0; i < 10; i++) {  String poducerMsg = "producer: " + i;  System.***out***.println("produced message : "+ poducerMsg);  blockingQueue.put(poducerMsg);  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  **class** Consumer **implements** Runnable {  **private** **final** BlockingQueue<String> blockingQueue;    **public** Consumer(BlockingQueue<String> queue) {  **this**.blockingQueue = queue;  }  @Override  **public** **void** run() {  **while**(**true**) {  **try** {  String consumeMsg = blockingQueue.take();  System.***out***.println("consumed message : "+ consumeMsg);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  } |

**Program: producer consumer design pattern using wait and notify methods**

|  |
| --- |
|  |

**Program: find the duplicate elements**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {    String str[] = {"ab","nag","java","scala","dig","ab","scala","java"};  **for**(**int** i=0; i<str.length; i++) {  **for**(**int** j=i+1; j<str.length; j++) {  **if**(str[i].equals(str[j])) {  System.***out***.println(str[i]);  }  }  }    }  }  Or  **public** **static** **void** main(String args[]) **throws** IOException {  String strArr[] = {"ab","nag","java","scala","dig","ab","scala","java"};  Set<String> set = **new** HashSet<>();  **for**(String str: strArr) {  **if**(!set.add(str)) {  System.***out***.println(str);  }  }  }  } |

**Program: find the largest and smallest number in array**

|  |
| --- |
| **public** **class** demo {  **public** **static** **void** main(String args[]) **throws** IOException {  **int** inputNumArry[] = { 10, 25, 5, 70, 80, 2, 87, 99 };    **int** maxNumber = inputNumArry[0];  **int** minNumber = inputNumArry[0];    **for** (**int** i = 0; i < inputNumArry.length; i++) {  System.***out***.println(inputNumArry[i]);  **if** (maxNumber < inputNumArry[i]) {  maxNumber = inputNumArry[i];  } **else** **if** (minNumber > inputNumArry[i]) {  minNumber = inputNumArry[i];  }  }  System.***out***.println("maximum number: " + maxNumber);  System.***out***.println("minmum number: " + minNumber);  }  } |

**Program: find the common element in two arrays**

|  |
| --- |
| **public** **class** demo {  **public** **static** **void** main(String args[]) **throws** IOException {  **int** firstArry[] = { 10, 25, 5, 70, 80, 2, 87, 99 };  **int** secondtArry[] = { 5, 80, 40, 100, 87, 99, 14, 100 };    **for**(**int** i=0; i< firstArry.length; i++) {  **for**(**int** j=0; j< secondtArry.length; j++) {  **if**(firstArry[i] == secondtArry[j]) {  System.***out***.println("common element: " + firstArry[i]);  **break**;  }  }  }  }  } |

**Program: remove the duplicate elements from array**

|  |
| --- |
| **public** **class** demo {  **public** **static** **void** main(String args[]) **throws** IOException {  **int** inputArr[] = { 10, 25, 5, 85, 2, 10, 85 };  **for** (**int** ele : inputArr) {  System.***out***.println("original array: " + ele);  }  **int** ouputArry[] = Arrays.*stream*(inputArr).distinct().toArray();  **for** (**int** ele : ouputArry) {  System.***out***.println("remove duplicate array: " + ele);  }  } |

**Program: find out firstlargest and secondlargest number in array**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  **int** inputNumArry[] = { 10, 25, 5, 70, 80, 2, 87, 99 };    **int** firstLargestNumber = 0;  **int** secondLargestNumber = 0;    **for** (**int** i = 0; i < inputNumArry.length; i++) {  System.***out***.println(inputNumArry[i]);  **if** (firstLargestNumber < inputNumArry[i]) {  secondLargestNumber = firstLargestNumber;  firstLargestNumber = inputNumArry[i];  } **else** **if** (secondLargestNumber > inputNumArry[i]) {  secondLargestNumber = inputNumArry[i];  }  }  System.***out***.println("firstLargestNumber number: " + firstLargestNumber);  System.***out***.println("secondLargestNumber number: " + secondLargestNumber);  } |

**Program: find factorial of given number with iteration**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  **int** num =5;  **int** fact = 1;  **for**(**int** i =1; i<=num; i++) {  //System.out.print(fact +"\*"+ i+"= ");  //System.out.print(fact\*i+",");  fact = fact\*i;  }  System.***out***.println(fact);  }  } |

**Program: find factorial of given number using recursion**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  System.***out***.println(*factorial*(5));  }  **static** **int** factorial(**int** num) {  **if** (num == 0)  **return** 1;  **else**  **return** num \* *factorial*(num - 1);  }  } |

**Program: reverse number**

|  |
| --- |
| **public** **static** **void** main(String[] args) {  //original number  **int** number = 1234;  **int** reversedNumber = 0;  **int** temp = 0;    **while**(number > 0){    //use modulus operator to strip off the last digit  temp = number%10;    //create the reversed nzzzzzzzumber  reversedNumber = reversedNumber \* 10 + temp;  number = number/10;  }  //output the reversed number  System.***out***.println("Reversed Number is: " + reversedNumber);  }  } |

**Program: Calculate the Power of a Number without using in-built library**

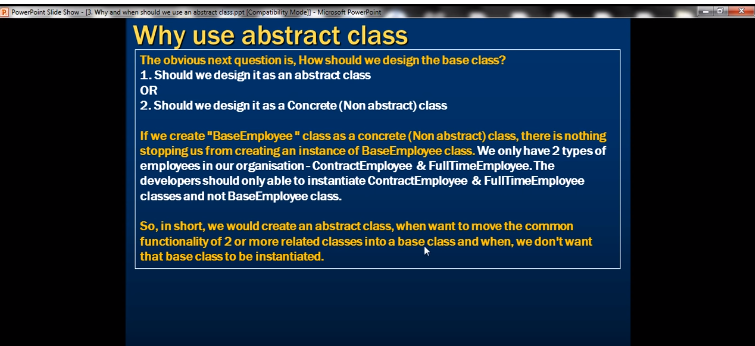
|  |
| --- |
| **public** **class** demo {  **public** **static** **void** main(String[] args) {  **int** basenumber = 2, exponent = 3;  **long** powerVal = 1;  **while** (exponent > 0) {  powerVal \*= basenumber;  --exponent;  }  System.***out***.println("Result: " + powerVal);  }  }  Or  double result = Math.pow(basenumber, exponent);  **public** **class** javaTrickOveloding {  **public** **static** **void** main(String args[]) {  *test*(**null**);  }  **public** **static** **void** test(Object obj) {  System.***out***.println("object impl called");  }    /\*public static void test(String obj) {  System.out.println("String impl called");  }\*/    **public** **static** **void** test(Integer obj) {  System.***out***.println("integer impl called");  }  }  Output:  integer impl called |

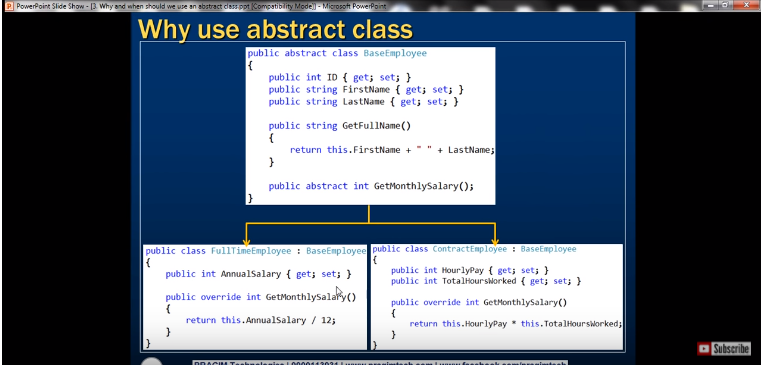
**Abstract class:**

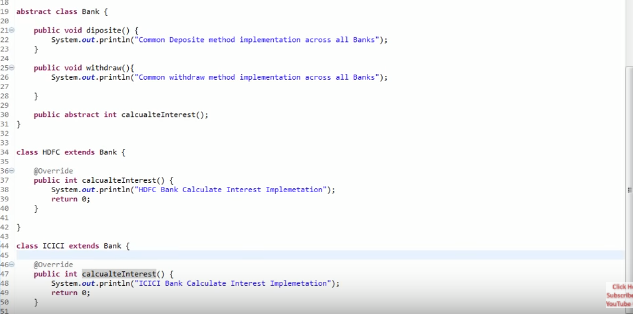
When want to create abstract class, when want to move common functionality of 2 or more related classes into base class and when we don’t want that base class to **be** instantiated.

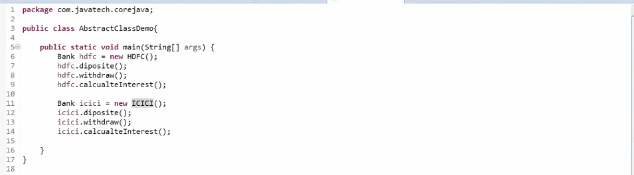
Example:

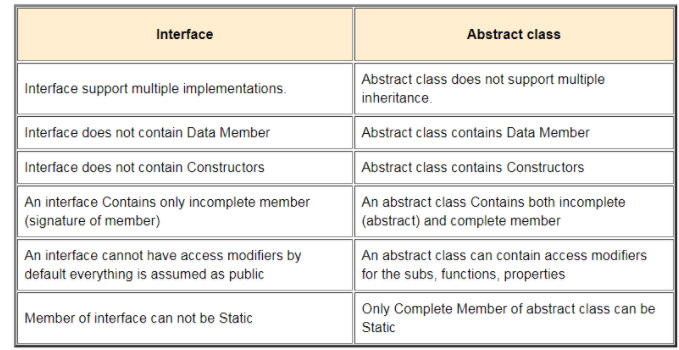
|  |
| --- |
| **Abstract Class:**  **public** **abstract** **class** BaseEmployee {  **private** **int** id;  **private** String firstName;  **private** String lastName;    **public** BaseEmployee(**int** id, String firstName, String lastName) {  **this**.id = id;  **this**.firstName = firstName;  **this**.lastName = lastName;  }  **public** String getFullName() {  **return** **this**.firstName +" "+ **this**.lastName;  }  **public** **abstract** **int** getMonthlySalary();  }  **SubClasses:**  **public** **class** ContractEmployee **extends** BaseEmployee {  **private** **int** hourlyPay;  **private** **int** totalHour;    **public** ContractEmployee(**int** id, String firstName, String lastName, **int** hourlyPay, **int** totalHour) {  **super**(id,firstName,lastName);  **this**.hourlyPay = hourlyPay;  **this**.totalHour = totalHour;  }  @Override  **public** **int** getMonthlySalary() {  **return** **this**.hourlyPay \* **this**.totalHour;  }  }  **public** **class** FullTimeEmployee **extends** BaseEmployee {  **private** **int** annualSalary;    **public** FullTimeEmployee(**int** id, String firstName, String lastName, **int** annualSalary) {  **super**(id,firstName,lastName);  **this**.annualSalary = annualSalary;  }  @Override  **public** **int** getMonthlySalary() {  **return** **this**.annualSalary / 12;  }  }  **MainRunner:**  **public** **class** EmployeeRunner {  **public** **static** **void** main(String[] args) {  BaseEmployee fullTimeEmp = **new** FullTimeEmployee(101,"Nagendra", "Mekala", 1000000);  System.***out***.println("Full time emp full name: "+ fullTimeEmp.getFullName());  System.***out***.println("Full time emp monthly salary: "+ fullTimeEmp.getMonthlySalary());    BaseEmployee contractEmp = **new** ContractEmployee(101,"James", "Goshlin", 500,20);  System.***out***.println("Contract time emp full name: "+ contractEmp.getFullName());  System.***out***.println("Contract emp monthly salary: "+ contractEmp.getMonthlySalary());    }  } |

****



****

****

****

If you are using JDK 8, there is no reason to use abstract classes because whatever we do with abstract classes we can now do it with interfaces because of default methods. If you use abstract class you have to extend it and there is a restriction that you can extends only once. But if you use interface you can implements as many as you want.

Interface is by default an abstract **class** an all methods and constructors are public.

# Spring Boot Exception Handling Using ControllerAdvice

**ResponseEntity**

ResponseEntity represents an HTTP response, including headers, body, and status. While @ResponseBody puts the return value into the body of the response, ResponseEntity also allows us to add headers and status code.

### Response Statuses for Errors

Use appropriate status code based on the error.

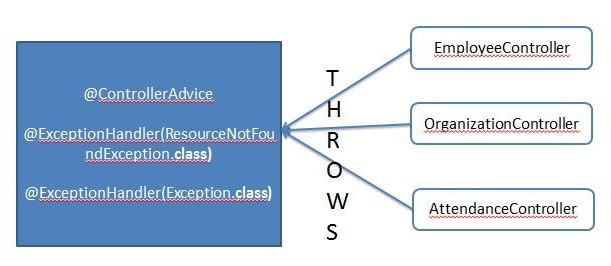
* 404 - RESOURCE NOT FOUND
* 400 - BAD REQUEST
* 401 - UNAUTHORIZED
* 415 - UNSUPPORTED TYPE - Representation not supported for the resource
* 500 - SERVER ERROR

**Let’s consider a few HTTP Methods:**

* GET : Should not update anything. Should be idempotent (same result in multiple calls). Possible Return Codes 200 (OK) + 404 (NOT FOUND) +400 (BAD REQUEST)
* POST : Should create new resource. Ideally return JSON with link to newly created resource. Same return codes as get possible. In addition - Return code 201 (CREATED) can be used.
* PUT : Update a known resource. ex: update client details. Possible Return Codes : 200(OK) + 404 (NOT FOUND) +400 (BAD REQUEST)
* DELETE : Used to delete a resource. Possible Return Codes : 200(OK).
* 500 - SERVER ERROR is possible with all the above HTTP methods. In the case of a 500, include the contact details of the help desk in the response.

Spring provides a very useful way to handle exceptions using ControllerAdvice.   
We will be implementing a ControlerAdvice class which will handle all exceptions thrown by the controller class.

Exceptions thrown by a Controller method is mapped to the ControllerAdvice method using @ExceptionHandler annotations.



|  |
| --- |
| **public** **interface** LocationService {  **public** List<Location> getAllLocations();  **public** Location getLocationWithNoEmployeeFound(String id);  **public** Location getLocationInternalServerError(String id) ;  }  **public** **class** ErrorResponse {  **private** **boolean** error;  **private** String message;  **public** **boolean** isError() {  **return** error;  }  **public** **void** setError(**boolean** error) {  **this**.error = error;  }  **public** String getMessage() {  **return** message;  }  **public** **void** setMessage(String message) {  **this**.message = message;  }  }  **public** **class** InternalServerException **extends** RuntimeException {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;    **public** InternalServerException() {  **super**();  }  **public** InternalServerException(String message) {  **super**(message);  }  }  **public** **class** ResourceNotFoundException **extends** RuntimeException {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **public** ResourceNotFoundException() {  **super**();  }  **public** ResourceNotFoundException(String message) {  **super**(message);  }  }  @Service  **public** **class** LocationServiceImpl **implements** LocationService {  @Override  **public** List<Location> getAllLocations() {  List<Location> list = Arrays.*asList*(**new** Location("101", "globel village", "india", "kanranatka"),  **new** Location("102", "kld", "india", "ap"),  **new** Location("103", "mphsis", "us", "us")  );  **return** list;  }  @Override  **public** Location getLocationWithNoEmployeeFound(String id) {  **return** **null**;  }    @Override  **public** Location getLocationInternalServerError(String id) {  **if**(id == **null**) {  **throw** **new** InternalServerException("Ineternall server exception throw by service class");  }  **return** **null**;  }  }  **public** **interface** LocationService {  **public** List<Location> getAllLocations();  **public** Location getLocationWithNoEmployeeFound(String id);  **public** Location getLocationInternalServerError(String id) ;  }  **Controller Class:**  @RequestMapping("location")  @RestController  **public** **class** LocationController {  @Autowired  **public** LocationService service;  @GetMapping("/getAll")  **public** ResponseEntity<List<Location>> getAllLocations() {  List<Location> list = service.getAllLocations();  ResponseEntity<List<Location>> response = **new** ResponseEntity<>(list, HttpStatus.***ACCEPTED***);  **return** response;  }  @GetMapping("/getNot")  **public** ResponseEntity<Location> getById() {  Location location = **null**;  **try** {  location = service.getLocationWithNoEmployeeFound("101");  **if** (location == **null**) {  **throw** **new** ResourceNotFoundException("Location resource not found exception");  }  } **catch** (InternalServerException e) {  **throw** **new** ResourceNotFoundException(e.getMessage());  }  **return** **new** ResponseEntity<>(location, HttpStatus.***OK***);  }  @GetMapping("/getEx")  **public** ResponseEntity<Location> getInternalServerException() {  Location location = **null**;  **try** {  location = service.getLocationInternalServerError(**null**);  } **catch** (InternalServerException e) {  **throw** **new** InternalServerException(e.getMessage());  }  **return** **new** ResponseEntity<>(location, HttpStatus.***ACCEPTED***);  }  }  **Handler Exception class:**  @ControllerAdvice  **public** **class** ExceptionHandlerControllerAdvice {  @ExceptionHandler(ResourceNotFoundException.**class**)  **public** ResponseEntity<ErrorResponse> handleResourceNotFound(ResourceNotFoundException ex) {  ErrorResponse error = **new** ErrorResponse();  error.setError(**true**);  error.setMessage(ex.getMessage());  **return** **new** ResponseEntity<>(error,HttpStatus.***NOT\_FOUND***);  }    @ExceptionHandler(InternalServerException.**class**)  **public** ResponseEntity<ErrorResponse> handleInternalServerException(InternalServerException ex) {  ErrorResponse error = **new** ErrorResponse();  error.setError(**true**);  error.setMessage(ex.getMessage());  **return** **new** ResponseEntity<>(error,HttpStatus.***INTERNAL\_SERVER\_ERROR***);  }  } |