[13:49, 1: Whatnis checked and unchecked exception. Give one class name

[13:49, 1: How to create custom exception

Steps to create a Custom Exception with an Example

CustomException class is the custom exception class this class is extending Exception class.

Create one local variable message to store the exception message locally in the class object.

We are passing a string argument to the constructor of the custom exception object. The constructor set the argument string to the private string message.

toString() method is used to print out the exception message.

We are simply throwing a CustomException using one try-catch block in the main method and observe how the string is passed while creating a custom exception. Inside the catch block, we are printing out the message.

Example

class CustomException extends Exception {

String message;

CustomException(String str) {

message = str;

}

public String toString() {

return ("Custom Exception Occurred : " + message);

}

}

public class MainException {

public static void main(String args[]) {

try {

throw new CustomException("This is a custom message");

} catch(CustomException e) {

System.out.println(e);

}

}

}

[13:50, 1: Whatnis classnotfoundexception and classdefaounderror

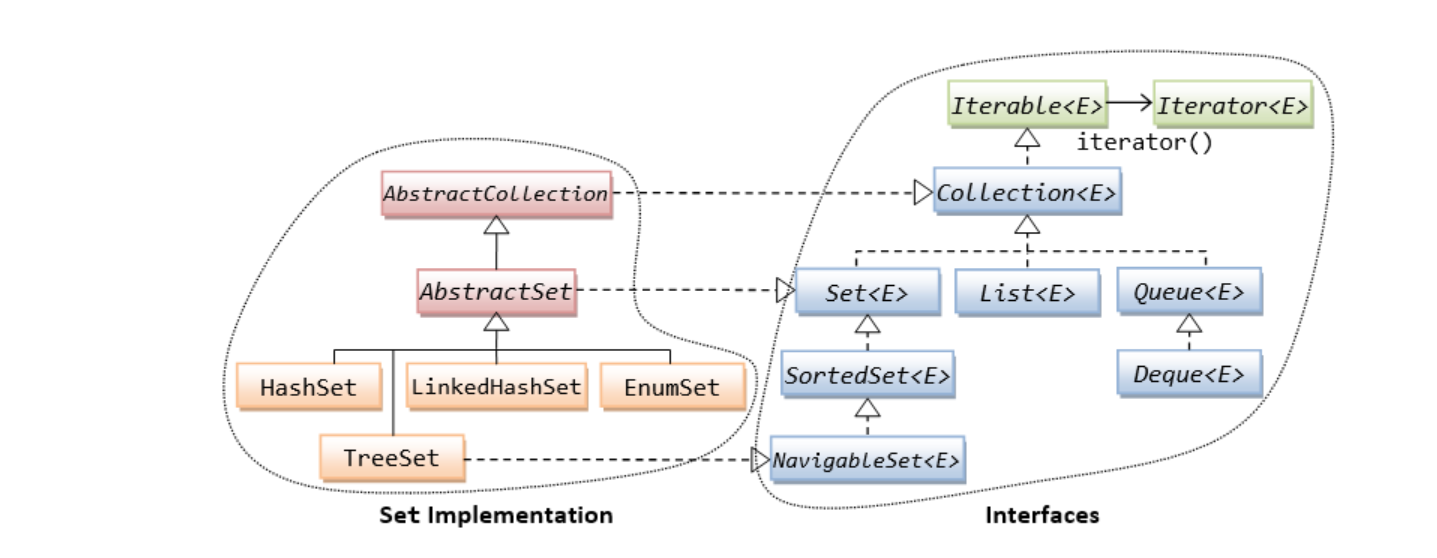
ClassNotFoundException

ClassNotFoundException is a runtime exception that is thrown when an application tries to load a class at runtime using the Class.forName() or loadClass() or findSystemClass() methods

NoClassDefFoundError

NoClassDefFoundError is an error that is thrown when the Java Runtime System tries to load the definition of a class, and that class definition is no longer available. The required class definition was present at compile time, but it was missing at runtime

[13:51, 1: List interface name and their implementation class in collection framework



[13:51, 1: Crate hassmap object and put some values and then iterate that map

Map<String, String> map = new HashMap<>();

map.put("one", "first");

map.put("two", "second");

map.put("three", "third");

Stream<String> stream = map.keySet().stream();

stream.forEach((value) -> {

System.out.println(value);

});

et<Map.Entry<String, String>> entries = map.entrySet();

Iterator<Map.Entry<String, String>> iterator =

entries.iterator();

while(iterator.hasNext()) {

Map.Entry<String, String> entry = iterator.next();

String key = entry.getKey();

String value = entry.getValue();

}

[13:51, 1: How to get only keys from hashmap

Map<String, String> map = new HashMap<>();

Iterator<String> iterator = map.keySet().iterator();

while(iterator.hasNext(){

String key = iterator.next();

String value = map.get(key);

}

[13:52, 1: What is role of entryset in hashmap

s used to create a set out of the same elements contained in the map. It basically returns a set view of the map or we can

 It basically returns a set view of the hash map

  HashMap<Integer, String> hash\_map = new HashMap<Integer, String>();

        // Mapping string values to int keys

        hash\_map.put(10, "Geeks");

        hash\_map.put(15, "4");

        hash\_map.put(20, "Geeks");

        hash\_map.put(25, "Welcomes");

        hash\_map.put(30, "You");

        // Displaying the HashMap

        System.out.println("Initial Mappings are: " + hash\_map);

        // Using entrySet() to get the set view

        System.out.println("The set is: " + hash\_map.entrySet());

    }

}

[13:52, 1: Class level locking and object level locking in thread

**1. Object level lock in Java**

**Object level lock** is mechanism when we want to synchronize a **non-static method** or **non-static code block** such that only one thread will be able to execute the code block on given instance of the class. This should always be done **to make instance level data thread safe**.

Object level locking can be done as below :

|  |
| --- |
|  |
| public class DemoClass  {      public synchronized void demoMethod(){}  }    or    public class DemoClass  {      public void demoMethod(){          synchronized (this)          {              //other thread safe code          }      }  } |

**Class level lock** prevents multiple threads to enter in synchronized block in any of all available instances of the class on runtime. This means if in runtime there are 100 instances of DemoClass, then only one thread will be able to execute demoMethod() in any one of instance at a time, and all other instances will be locked for other threads.

Class level locking should always be done **to make static data thread safe**. As we know that [**static**](https://howtodoinjava.com/java/keywords/java-static-keyword/) keyword associate data of methods to class level, so use locking at static fields or methods to make it on class level.

|  |
| --- |
| Various ways for class level locking |
| public class DemoClass  {      //Method is static      public synchronized static void demoMethod(){        }  } |

[13:53, 1: Which is good to use synchronized method or synchronized block

) Synchronized block can throw throw [java.lang.NullPointerException](http://java67.blogspot.sg/2012/09/what-is-nullpointerexception-in-java.html) if expression provided to block as parameter evaluates to null, which is not the case with synchronized methods.

4) In case of synchronized method, lock is acquired by thread when it enter method and released when it leaves method, either normally or by throwing Exception. On the other hand in case of synchronized block, thread acquires lock when they enter synchronized block and release when they leave synchronized block.

Read more: <https://www.java67.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html#ixzz6jOdiGl7O>

[13:54, 1: How to create immutable class

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables.
4. Make all **mutable fields final** so that its value can be assigned only once.
5. Initialize all the fields via a [constructor](https://www.journaldev.com/18899/constructor-in-java) performing deep copy.
6. Perform [cloning](https://www.journaldev.com/60/java-clone-object-cloning-java) of objects in the getter methods to return a copy rather than returning the actual object reference.

To understand points 4 and 5, let’s run the sample Final class that works well and values don’t get altered after instantiation.

FinalClassExample.java

package com.journaldev.java;

import java.util.HashMap;

import java.util.Iterator;

public final class FinalClassExample {

private final int id;

private final String name;

private final HashMap<String,String> testMap;

public int getId() {

return id;

}

public String getName() {

return name;

}

/\*\*

\* Accessor function for mutable objects

\*/

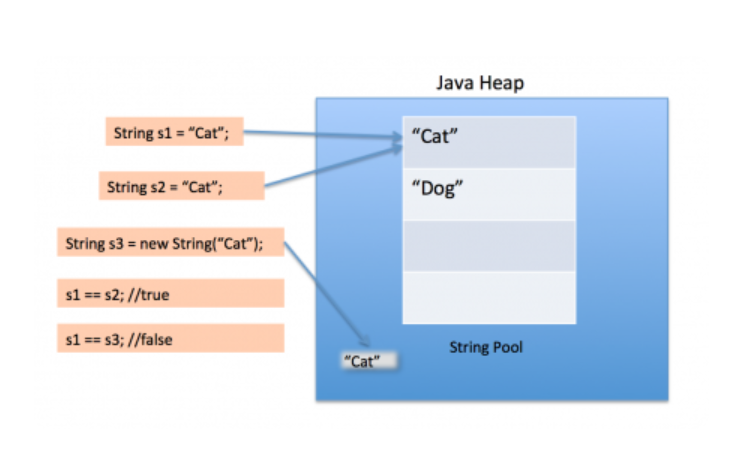
public HashMap<String, String> getTestMap() {

//return testMap;

return (HashMap<String, String>) testMap.clone();

}

[13:54, 1: String and string buffer differences. Wen to use which one



public static void main(String[] args) { String s1 = "Cat"; String s2 = "Cat"; String s3 = new String("Cat"); System.out.println("s1 == s2 :"+(s1==s2)); System.out.println("s1 == s3 :"+(s1==s3));

s1 == s2 :true s1 == s3 :false

[13:56, 1: String practical question

String st1 = new String("as");

String St2 = S1

Sop(st2);

[13:56, 1: Like this

[13:56, 1: 5 practical question from above pattern

[13:57, 1: What is new java memory model in Java 8

### Memory Management in Java – Java Heap Memory Switches

Java provides a lot of memory switches that we can use to set the memory sizes and their ratios. Some of the commonly used memory switches are:

|  |  |
| --- | --- |
| VM Switch | VM Switch Description |
| -Xms | For setting the initial heap size when JVM starts |
| -Xmx | For setting the maximum heap size. |
| -Xmn | For setting the size of the Young Generation, rest of the space goes for Old Generation. |
| -XX:PermGen | For setting the initial size of the Permanent Generation memory |
| -XX:MaxPermGen | For setting the maximum size of Perm Gen |
| -XX:SurvivorRatio | For providing ratio of Eden space and Survivor Space, for example if Young Generation size is 10m and VM switch is -XX:SurvivorRatio=2 then 5m will be reserved for Eden Space and 2.5m each for both the Survivor spaces. The default value is 8. |
| -XX:NewRatio | For providing ratio of old/new generation sizes. The default value is 2. |

Most of the times, above options are sufficient, but if you want to check out other options too then please check [JVM Options Official Page](https://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html).

### Memory Management in Java – Java Garbage Collection

Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space to be allocated to objects created in future processing. One of the best features of Java programming language is the **automatic garbage collection**, unlike other programming languages such as C where memory allocation and deallocation is a manual process.

**Garbage Collector** is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program. All these unreferenced objects are deleted and space is reclaimed for allocation to other objects.

One of the basic ways of garbage collection involves three steps:

1. **Marking**: This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
2. **Normal Deletion**: Garbage Collector removes the unused objects and reclaim the free space to be allocated to other objects.
3. **Deletion with Compacting**: For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.

[13:57, 1: What is streaming in java 8

tream provides following features:

* Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
* Stream is functional in nature. Operations performed on a stream does not modify it's source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
* Stream is lazy and evaluates code only when required.
* The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

[13:58, 1: How to use streaming with lamda

we can create a stream from individual objects using *Stream.of()*:

Stream.of(arrayOfEmps[0], arrayOfEmps[1], arrayOfEmps[2]);

[13:59, 1: Spring basic annotations use in spring mvc

1. @Controller Annotation

This annotation serves as a specialization of *@Component*, allowing for implementation classes autodetected through classpath scanning.*@Controller annotation* tells the Spring IOC container to treat this class as [Spring MVC controller](https://www.javadevjournal.com/spring-mvc/spring-controllers/).

To configure/ customize MVC components, *Spring MVC* provides an option to handle it either through *Java Config or XML*. Add @EnableWebMvc will import the Spring MVC configuration from.WebMvcConfigurationSupport For XML based configuration use the

<context:component-scan base-package="com.javadevjournal">

Copy

@Controller

public class SpringMVCController {

//HTTP Mappings

}

Copy

2. @RestController Annotation

A convenience annotation that is itself annotated with *@Controller* and *@ResponseBody*.

@RestController

public class FilterExampleController {

@GetMapping

public String greeting() {

return "Hello World";

}

@GetMapping(value = "/greeting")

public String customGreetings() {

return "Hello From Custom Greetings";

}

}

Copy

3. @RequestMapping

Annotation for mapping web requests methods in the *Spring MVC Controller*. Both *Spring MVC and Spring WebFlux* support this annotation.*@RequestMapping* annotation provides several options to customize its behavior.

* Consumes – The consumable media types of the mapped request, narrowing the primary mapping. (e.g. @RequestMapping(consumes = {"application/json", "application/xml"})).
* method – The HTTP request methods to map (e.g. method = {RequestMethod.GET,RequestMethod.POST}).
* header – The headers of the mapped request.
* name – the name of the mapping.
* value – The primary mapping expressed by this annotation
* produces – The producible media types of the mapped request.

Here is an example for the *@RequestMapping*

@Controller

public class SpringMVCController {

@RequestMapping(value = {

"/greetings",

"/hello-world"}, method = {RequestMethod.GET,RequestMethod.POST},

consumes = {"application/json","application/xml"},

produces = { "application/json"},headers = {"application/json"

})

public String hellpWorld() {

return "Hello";

}

}

Copy

[pullquote align=”normal”]This annotation can be used both at the class and at the method level [/pullquote]

@Controller

@RequestMapping(value = {

"/greetings"

}, method = {

RequestMethod.GET

})

public class SpringMVCController {}

Copy

If you are using *Spring 4.3+*, check out new and improved [@RequestMapping annotations](https://www.javadevjournal.com/spring-mvc/spring-new-requestmapping-annotations/).

4. @RequestParam

Annotation which shows that it binds a method parameter to a web request parameter. Request parameters passed by the browser/client as part of the HTTP request, the @RequestParam annotation help to map these parameters easily at the controller level.

@GetMapping("/request-mapping-example")

public String requestMappingExample(@RequestParam("code") String code) {

//

}

Copy

With @RequestParam we can specify default value when Spring finds no or empty value in the request.

public String requestMappingExample(@RequestParam(defaultValue = "1") long productQty){}

Copy

5. @PathVariable

This annotation shows that a method parameter bound to a URI template variable. We specify the variable as part of the @RequestMapping and bind a method argument with *@PathVariable*. Let’s take an example where we want to pass productCode as part of the URI and not request parameter.

@GetMapping("/products/{id}")

public String getProduct(@PathVariable("id") String id) {

//

}

Copy

Please note following extra points while using *@PathVariable*.

* The variable name in the @PathVariable annotation is optional if the name of the part in the template matches the name of the method argument. For the above example, we can omit “id” from the @PathVariable annotation.
* @GetMapping("/products/{id}")
* public String getProduct(@PathVariable String id) {
* //

}

Copy

* Use “require” parameter to mark the path variable as an optional.
* @GetMapping("/products/{id}")
* public String getProduct(@PathVariable(required = false) String id) {
* //

}

Copy

Here, id is an optional field

6. @SessionAttribute

Annotation to bind a method parameter to a session attribute.[@SessionAttribute](https://www.javadevjournal.com/spring-mvc/spring-mvc-session-attributes/) used to pass value across different requests through the session. Rather than using HttpSession object directly, using this annotation can benefit auto type conversion and optional/required check.

@GetMapping("/user")

public String sessionexample(@SessionAttribute(name = "userLoginTime") LocalDateTime startDateTime) {

//

}

Copy

7. @RequestBody

The [@RequestBody annotation](https://www.javadevjournal.com/spring/spring-request-response-body/) showing a method parameter bound to the body of the web request. It passes the body of the request through an HttpMessageConverter to resolve the method argument depending on the content the request.

@PostMapping("/product/save") public String saveProduct(@RequestBody Product product){}

Copy

8. @ResponseBody

The [@ResponseBody Annotation](https://www.javadevjournal.com/spring/spring-request-response-body/) that shows a method return value bound to the web response body. Supported for annotated handler methods.**Spring treats the result of the method as the response itself.**

@GetMapping("/products/{id}")

public @ResponseBody Product saveProduct(@PathVariable("id") String id) {

//

}

Copy

9. @ExceptionHandler

[ExceptionHandler](https://www.javadevjournal.com/spring/exception-handling-for-rest-with-spring/) is a *Spring annotation* handle exceptions thrown by request handling. This annotation works at the @Controller level.

@GetMapping("/greeting")

String greeting() throws Exception {

//

}

@ExceptionHandler({

Exception.class

})

public handleException() {

//

}

Copy

10. @InitBinder

Annotation that identifies methods which initialize the *WebDataBinder* and used for populating command and form object arguments of *annotated handler* methods.

@InitBinder

protected void initBinder(WebDataBinder binder) {

binder.setValidator(customValidator);

}

Copy

11. @ModelAttribute

@ModelAttribute refers to a property of the Model object in *Spring MVC*. This [ModelAttribute annotation](https://www.javadevjournal.com/spring-mvc/spring-mvc-model-attribute-annotation/" \o "Spring MVC @ModelAttribute Annotation" \t "_blank) binds a method parameter or method return value to a named model attribute, exposed to a web view.

@PostMapping("/customer-registration")

public String processForm(@ModelAttribute("customer") Customer customer) {}

Copy

The annotation is used to define objects which should be part of a Model. So if you want to have a Customer object referenced in the Model you can use the following method:

@ModelAttribute("customer")

public Person getCustomer() {

//

}

Copy

we don’t have to specify the model key, Spring uses the method’s name by default

@ModelAttribute

public Person getCustomer() {

//

}

Copy

12. @Qualifier Annotation

The ***@Qualifier annotation*** helps disambiguate bean references when Spring otherwise could not do so. In our case, to we can use the *@Qualifier annotation* help in the issue to choose the correct bean for the dependency injection. Let’s change our previous example:

public class OrderFacade {

@Qualifier("userServiceImpl")

@Autowired

private UserService userService;

}

Copy

For more details please read [@Qualifier Annotation](https://www.javadevjournal.com/spring/qualifier-annotation/)

13. @CrossOrigin

This annotation allows the cross-domain communication for the annotated handler methods. This @CrossOrigin annotation enables cross-origin resource sharing only for this specific method. Let’s take an example where we want to allow only [http://localhost:9002](http://localhost:9000/) to send cross-origin requests.

@CrossOrigin(origins = "http://localhost:9002")

@GetMapping("/hello")

public String greeting(@RequestParam(required = false, defaultValue = "Stranger") String name) {

return "Hello" + name;

}

[14:00, 1: How to make rest controller from simple controller

@Controller public class HomeController { @GetMapping("/hello") public String home(Locale locale, Model model) { Date date = new Date(); DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale); String formattedDate = dateFormat.format(date); model.addAttribute("serverTime", formattedDate); return "home"; } }

[14:00, 1: What is modelandview

[14:01, 1: What is @Required in spring

[14:02, 1: How spring transaction mangement works?

**Connection** connection = dataSource.getConnection(); // **(1)**

**try** (connection) {

connection.setAutoCommit(**false**); // **(2)**

*// execute some SQL statements...*

connection.commit(); // **(3)**

} **catch** (**SQLException** e) {

connection.rollback(); // **(4)**

}

[14:03, 1: What is "new" "required" "support" in transaction

When looking at the Spring source code, you’ll find a variety of propagation levels or modes that you can plug into the @Transactional method.

@Transactional(propagation = Propagation.REQUIRED)

*// or*

@Transactional(propagation = Propagation.REQUIRES\_NEW)

[14:03, 1: Hibernate configuration and all the annotations used in Hibernate

1. <hibernate-configuration>
2. <session-factory>
4. <property name="hbm2ddl.auto">update</property>
5. <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
6. <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
7. <property name="connection.username">system</property>
8. <property name="connection.password">jtp</property>
9. <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>
11. <mapping **class**="com.javatpoint.mypackage.Employee"/>
12. </session-factory>
13. </hibernate-configuration>

[14:04, 1: Cascade in hibernate

*<!-- Stock.hbm.xml -->*

<set name="stockDailyRecords" cascade="save-update" table="stock\_daily\_record"...>

<key>

<column name="STOCK\_ID" not-null="true" />

</key>

<one-to-many class="com.mkyong.common.StockDailyRecord" />

</set>

<set name="stockDailyRecords" cascade="delete" table="stock\_daily\_record"

[14:04, 1: Whatb is auto sequence in Hibernate

<hibernate-mapping>

<class name="com.tech.spring4.model.User" table="Customer">

<id name="id" type="long">

<column name="USERID" unique="true"/>

<generator class="increment"/>

</id>

<property name="username"><column name="username" length="30" not-null="true"></column></property>

<property name="email"><column name="email" length="100" not-null="true"></column></property>

<property name="address"><column name="address" length="100" not-null="true"></column></property>

</class>

</hibernate-mapping>

[14:05, 1: Yahi sab tha

[14:05, 1: Basic jada puch raha tha

[14:05, 1: Microservices ka kar lijiyega

[14:05, 1: How to create spring boot project

[14:06, 1: What us profile in spring boot

[14:06, 1: How to run spring boot project with yml configuration