What is spring boot ?

Spring Boot is **an open source, microservice-based Java web framework**. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase.

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Advantages of Spring boot ?

- Fast and easy development of Spring-based applications;
- No need for the deployment of war files;
- The ability to create standalone applications;
- Helping to directly embed Tomcat, Jetty, or Undertow into an application;
- No need for XML configuration;
- Reduced amounts of source code;

diff between @restcontroller and @controller annotations?

@Controller annotation indicates that the class is a “controller” like a web controller. @RestController annotation indicates that class is a controller where @RequestMapping methods assume @ResponseBody semantics by default. In @Controller, we need to use @ResponseBody on every handler method.

@Controller	@RestController

@Controller	@RestController
@Controller is used to mark classes as Spring MVC Controller.	@RestController annotation is a special controller used in RESTful Web services, and it's the combination of @Controller and @ResponseBody annotation.
It is a specialized version of @Component annotation.	It is a specialized version of @Controller annotation.
In @Controller, we can return a view in Spring Web MVC.	In @RestController, we can not return a view.
@Controller annotation indicates that the class is a "controller" like a web controller.	@RestController annotation indicates that class is a controller where @RequestMapping methods assume @ResponseBody semantics by default.
In @Controller, we need to use @ResponseBody on every handler method.	In @RestController, we don't need to use @ResponseBody on every handler method.
It was added to Spring 2.5 version.	It was added to Spring 4.0 version.

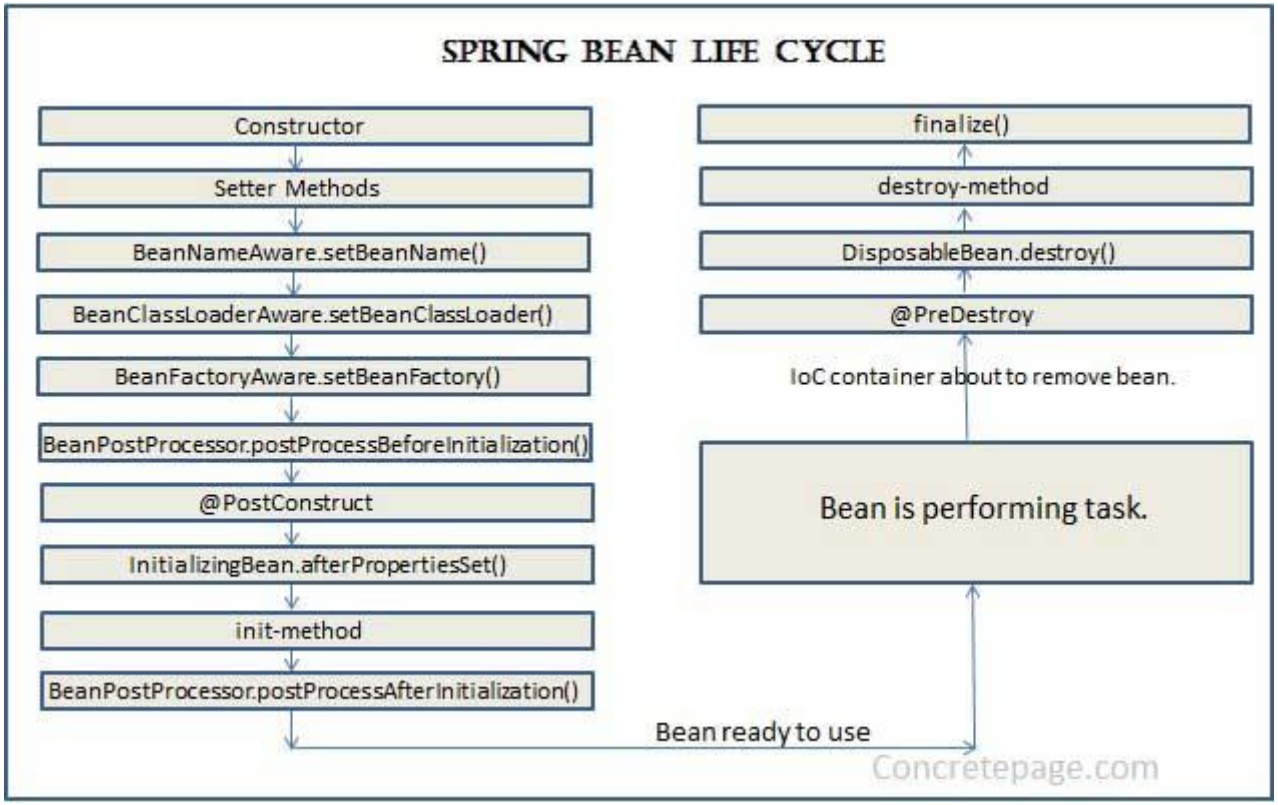
Autowiring and its Types?

Autowiring feature of spring framework **enables you to inject the object dependency implicitly**. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only.

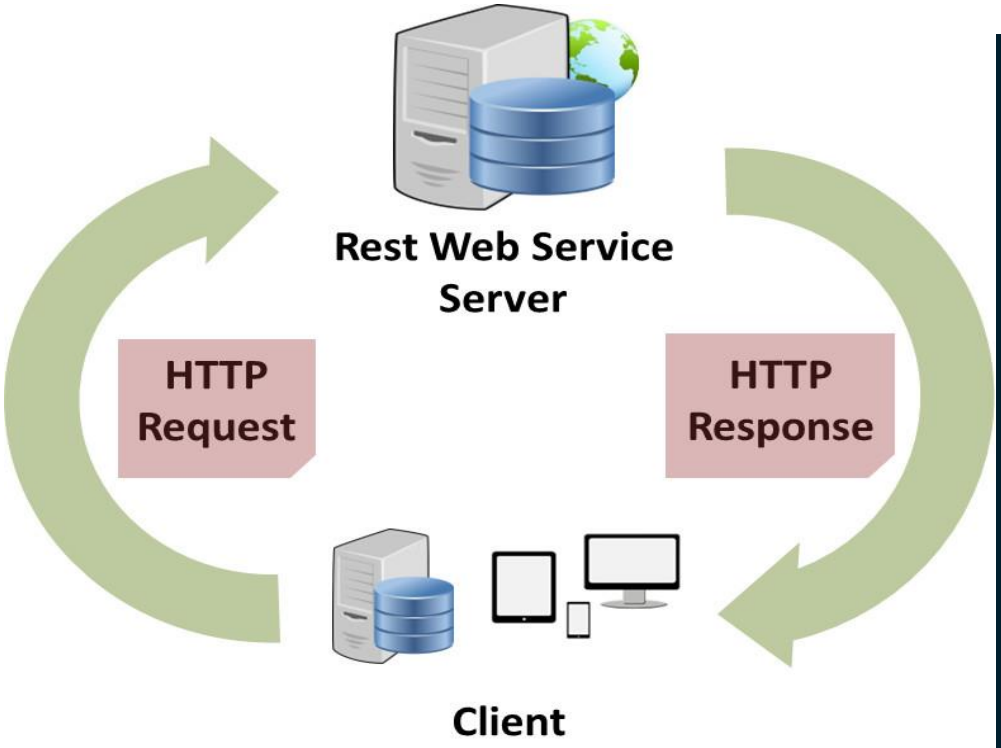
1. 'no' means the autowiring is OFF.
2. 'byName' will look for a bean named exactly the same as the property that needs to be autowired.
3. 'byType' permits a property to be autowired if there is exactly one bean of the property type in the container.
4. 'constructor' is equivalent to byType but operates to constructor arguments

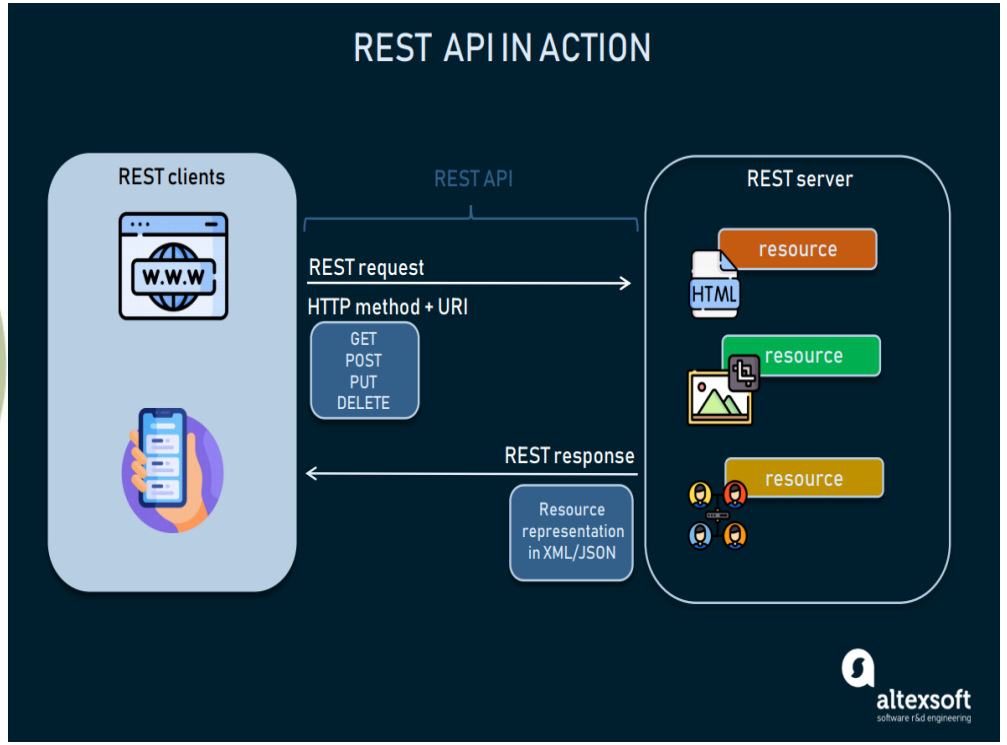
5. ‘Autodetect’ has been deprecated.

Spring bean life cycle?



Steps to create rest API ?





How to Design a REST API

1. Identify the resources – Object Modeling. The very first step in designing a REST API-based application is – identifying the objects which will be presented as resources. ...
2. Create Model URIs. ...
3. Determine Resource Representations. ...
4. Assigning HTTP Methods. ...
5. More Actions.

Get, post, put, delete and patch method?

Using HTTP Methods for RESTful Services

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	**CRUD**	**Entire Collection (e.g. /customers)**	**Specific Item (e.g. /customers/{id})**
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource alread
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), i found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), i found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Diff bet put and patch?

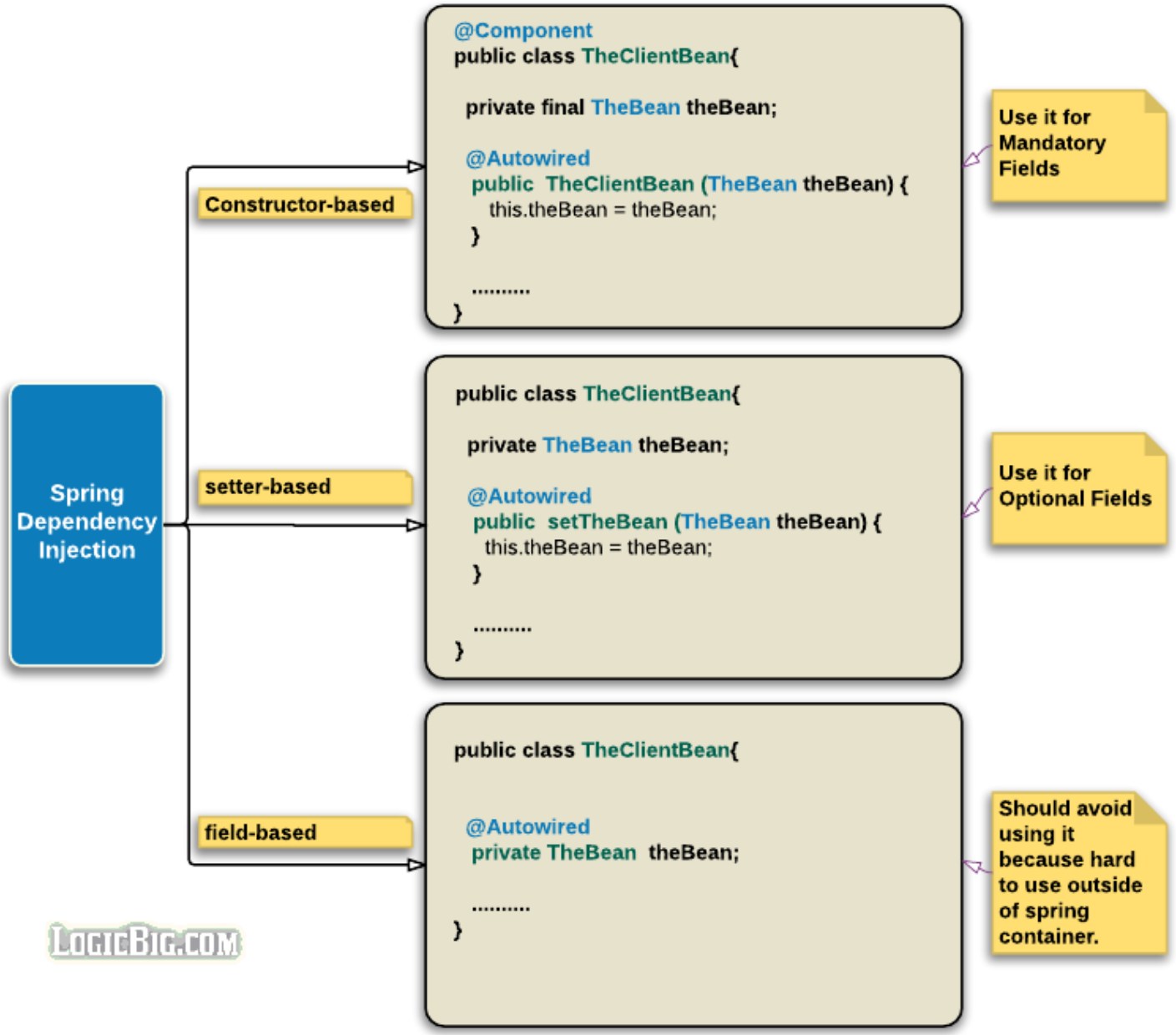
PUT	PATCH
PUT is a method of modifying resource where the client sends data that updates the entire resource .	PATCH is a method of modifying resources where the client sends partial data that is to be updated without modifying the entire data.
In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced	With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.
HTTP PUT is said to be idempotent, So if you send retry a request multiple times, that should be equivalent to a single request modification	HTTP PATCH is basically said to be non-idempotent. So if you retry the request N times, you will end up having N resources with N different URIs created on the server.
It has High Bandwidth	Since Only data that need to be modified if send in the request body as a payload , It has Low Bandwidth

What is IOC ?

Spring IoC Container is **the core of Spring Framework**. It creates the objects, configures and assembles their dependencies, manages their entire life cycle. The Container uses Dependency Injection(DI) to manage the components that make

Dependency Injection is **a fundamental aspect of the Spring framework, through which the Spring container "injects" objects into other objects or "dependencies"**. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Different ways of DI in Spring



Constructor injection Vs. Setter injection

Setter Injection	Constructor Injection
In Setter Injection, partial injection of dependencies can possible, means if we have 3 dependencies like int, string, long, then its not necessary to inject all values if we use setter injection. If you are not inject it will takes default values for those primitives.	In constructor injection, partial injection of dependencies cannot possible, because for calling constructor we must pass all the arguments, if not so we may get error.
Setter Injection will overrides the constructor injection value, provided if we write setter and constructor injection for the same property	But, constructor injection cannot overrides the setter injected values
We can use Setter injection when a number of dependencies are more or you need readability.	If readability is not a concern then we can use Constructor injection when a number of dependencies are more.
Setter injection makes bean class object as mutable [We can change]	Constructor injection makes bean class object as immutable [We cannot change]

What is bean and scope of bean ?

Bean Scopes refers to the lifecycle of Bean that means when the object of Bean will be instantiated, how long does that object live, and how many objects will be created for that bean throughout. Basically, it controls the instance creation of the bean and it is managed by the spring container. Bean Scopes in Spring.

Spring Bean Scopes

singleton ⇒ only one instance of bean per spring container(default)

prototype ⇒ a new instance every time a bean is requested

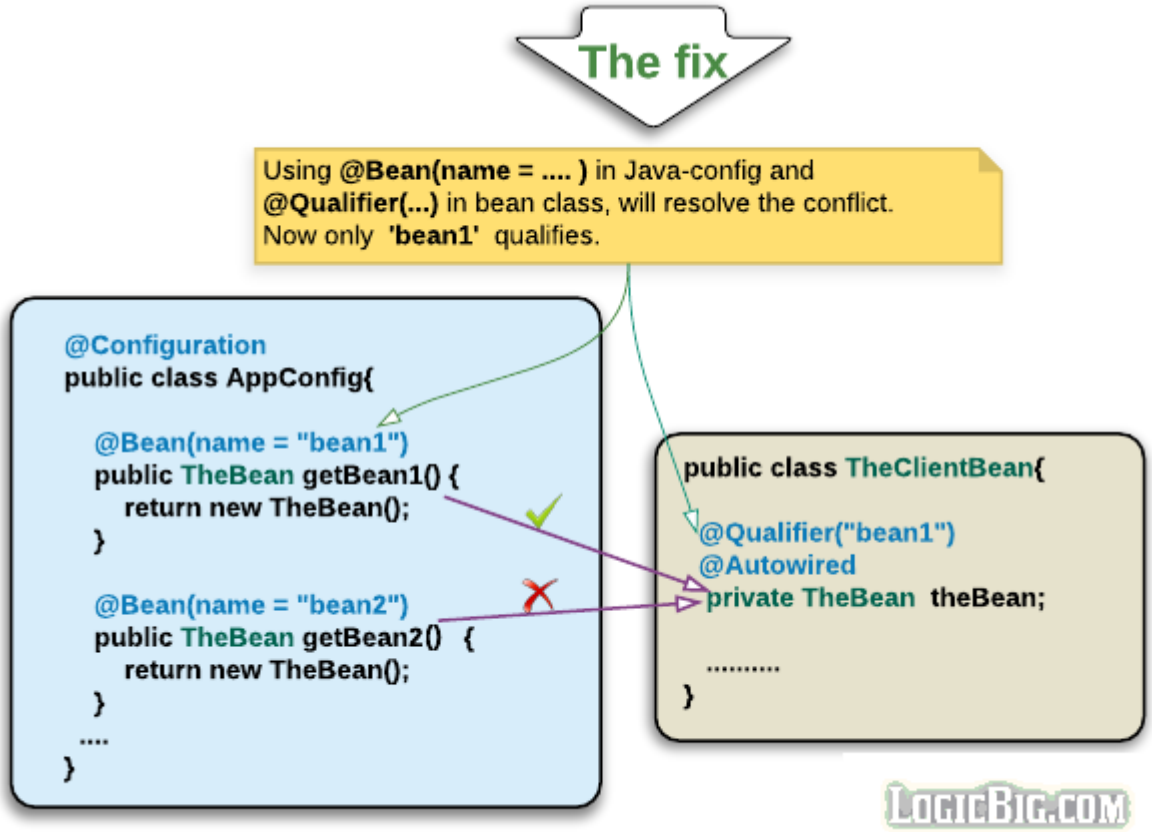
request ⇒ Single bean instance per HTTP request

session ⇒ Single bean instance per HTTP session

global-session ⇒ Single bean instance per global HTTP session

@Qualifier vs @Primary

There's another annotation called @Primary that we can use to decide which bean to inject when ambiguity is present regarding dependency injection. This annotation defines a preference when multiple beans of the same type are present.



Exception handling in spring boot ?

Handling exceptions and errors in APIs and sending the proper response to the client is good for enterprise applications. In this chapter, we will learn how to handle exceptions in Spring Boot.

Before proceeding with exception handling, let us gain an understanding on the following annotations.

Controller Advice

The @ControllerAdvice is an annotation, to handle the exceptions globally.

Exception Handler

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

You can use the following code to create @ControllerAdvice class to handle the exceptions globally −

```
package com.tutorialspoint.demo.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;

@ControllerAdvice
  public class ProductExceptionController {
}
```

Define a class that extends the RuntimeException class.

```
package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {
  private static final long serialVersionUID = 1L;
}
```

You can define the @ExceptionHandler method to handle the exceptions as shown. This method should be used for writing the Controller Advice class file.

```
@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {
}
```

Now, use the code given below to throw the exception from the API.

```
@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
public ResponseEntity<Object> updateProduct() {
  throw new ProductNotfoundException();
}
```

The complete code to handle the exception is given below. In this example, we used the PUT API to update the product. Here, while updating the product, if the product is not found, then return the response error message as “Product not found”. Note that the **ProductNotFoundException** exception class should extend the **RuntimeException**.

```
package com.tutorialspoint.demo.exception;
```

```
public class ProductNotfoundException extends RuntimeException {
  private static final long serialVersionUID = 1L;
}
```

The Controller Advice class to handle the exception globally is given below. We can define any Exception Handler methods in this class file.

```
package com.tutorialspoint.demo.exception;
```

Spring MVC annotations?

Diff bet spring MVC and Spring Boot?

Spring Boot	Spring MVC
Spring Boot is a module of Spring for packaging the Spring-based application with sensible defaults.	**Spring MVC** is a model view controller-based web framework under the Spring framework.
It provides default configurations to build **Spring-powered** framework.	It provides **ready to use** features for building a web application.
There is no need to build configuration manually.	It requires build configuration manually.
There is **no requirement** for a deployment descriptor.	A Deployment descriptor is **required**.
It avoids boilerplate code and wraps dependencies together in a single unit.	It specifies each dependency separately.
It **reduces** development time and increases productivity.	It takes **more** time to achieve the same.

spring	Spring Boot
Spring is the most popular Java EE framework for app building.	Spring Boot is a widely used framework for building REST APIs.
The main goal is to simplify Java EE development. However, Spring presupposes some boilerplate code to execute the minimal task.	The main goal is to deliver the easiest way of web app development and shorten the code length.
Dependency Injection (DI) and **Inversion of Control** (IOC) are the primary features of the Spring framework. Excellent tool to build loosely coupled apps.	**Spring Boot Actuator** delivers production-level features including app monitoring and metrics.
Spring is light-weight, supports the **MVC**, **WebFlux**, **ORM**, and **AOP** modules.	**Easy dependency management**. Spring Boot packages the compatible third-party dependencies for every Spring app type and delivers them to developers via starters.
Dependency management. Developers define the dependencies for the project manually in pom.xml.	**Autoconfiguration**. After choosing the starter, Spring Boot configures the app base on the jar dependencies automatically.
JDBC abstraction layer ensures an exception hierarchy, making error-handling easier.	**Embedded servlet container support**.
Spring supports all types and aspects of app development: web apps, core Java, enterprise apps, distributed apps.	**Spring Initializr**. Using this tool, it's possible to generate a skeleton of a Spring Boot project with any functionality required.
Requirements: Java 8+, Spring 5.0 and above, Servlet 3.0+ container **Testing**: TestContext, Spring MVC Test, mock objects, WebTestClient **Languages**: Groovy, Kotlin, dynamic languages	

Spring boot annotations?

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide **supplemental** information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

In this section, we are going to discuss some important **Spring Boot Annotation** that we will use later in this tutorial.

1 **@Required:** It applies to the **bean** setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception **BeanInitilizationException**.

2 @Autowired: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

3 @Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

4 @ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

5 @Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

Spring Framework Stereotype Annotations

@Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with **@Component** is found during the classpath. The Spring Framework pick it up and configure it in the application context as a **Spring Bean**.

@Controller: The @Controller is a class-level annotation. It is a specialization of **@Component**. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping** annotation

@Service: It is also used at class level. It tells the Spring that class contains the **business logic**.

@Repository: It is a class-level annotation. The repository is a **DAOs** (Data Access Object) that access the database directly. The repository does all the operations related to the database.

Spring Boot Annotations

- **@EnableAutoConfiguration:** It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication**.
- **@SpringBootApplication:** It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan,** and **@Configuration**.

Spring MVC and REST Annotations

- **@RequestMapping:** It is used to map the **web requests**. It has many optional elements like **consumes, header, method, name, params, path, produces**, and **value**. We use it with the class as well as the method.
- **@GetMapping:** It maps the **HTTP GET** requests on the specific handler method. It is used to create a web service endpoint that **fetches** It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**
- **@PostMapping:** It maps the **HTTP POST** requests on the specific handler method. It is used to create a web service endpoint that **creates** It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**
- **@PutMapping:** It maps the **HTTP PUT** requests on the specific handler method. It is used to create a web service endpoint that **creates** or **updates** It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**
- **@DeleteMapping:** It maps the **HTTP DELETE** requests on the specific handler method. It is used to create a web service endpoint that **deletes** a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**
- **@PatchMapping:** It maps the **HTTP PATCH** requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**
- **@RequestBody:** It is used to **bind** HTTP request with an object in a method parameter. Internally it uses **HTTP MessageConverters** to convert the body of the request. When we annotate a method parameter with **@RequestBody,** the Spring framework binds the incoming HTTP request body to that parameter.
- **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.
- **@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

- **@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a **query parameter**. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
- **@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a **method parameter**. The optional elements of the annotation are **name, required, value, defaultValue.** For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
- **@RestController:** It can be considered as a combination of **@Controller** and **@ResponseBody** annotations**.** The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.
- **@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

Main Class	@SpringBootApplication	Spring Boot auto configuration
REST Endpoint	@RestController	Class with REST endpoints
	@RequestMapping	REST endpoint method
	@PathVariable	URI path parameter
	@RequestBody	HTTP request body
Periodic Tasks	@Scheduled	Method to run periodically
	@EnableScheduling	Enable Spring's task scheduling
Beans	@Configuration	A class containing Spring beans
	@Bean	Objects to be used by Spring IoC for dependency injection
Spring Managed Components	@Component	A candidate for dependency injection
	@Service	Like @Component
	@Repository	Like @Component, for data base access
Persistence	@Entity	A class which can be stored in the data base via ORM
	@Id	Primary key
	@GeneratedValue	Generation strategy of primary key
	@EnableJpaRepositories	Triggers the search for classes with @Repository annotation
	@EnableTransactionManagement	Enable Spring's DB transaction management through @Beans objects
Miscellaneous	@Autowired	Force dependency injection
	@ConfigurationProperties	Import settings from properties file
Testing	@SpringBootTest	Spring integration test
	@AutoConfigureMockMvc	Configure MockMvc object to test HTTP queries

-

No.	SOAP	REST
1)	SOAP is a **protocol**.	REST is an **architectural style**.
2)	SOAP stands for **Simple Object Access Protocol**.	REST stands for **REpresentational State Transfer**.
3)	SOAP **can't use REST** because it is a protocol.	REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
4)	SOAP **uses services interfaces to expose the business logic**.	REST **uses URI to expose business logic**.
5)	**JAX-WS** is the java API for SOAP web services.	**JAX-RS** is the java API for RESTful web services.
6)	SOAP **defines standards** to be strictly followed.	REST does not define too much standards like SOAP.
7)	SOAP **requires more bandwidth** and resource than REST.	REST **requires less bandwidth** and resource than SOAP.
8)	SOAP **defines its own security**.	RESTful web services **inherits security measures** from the underlying transport.
9)	SOAP **permits XML** data format only.	REST **permits different** data format such as Plain text, HTML, XML, JSON etc.
10)	SOAP is **less preferred** than REST.	REST **more preferred** than SOAP.

-

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based ...

What is Spring Security used for?

Spring Security is the primary choice for **implementing application-level security in Spring applications**. Generally, its purpose is to offer you a highly customizable way of implementing authentication, authorization, and protection against common attacks.

What is Spring Security and how does it work?

At its core, Spring Security is really just **a bunch of servlet filters that help you add authentication and authorization to your web application**. It also integrates well with frameworks like Spring Web MVC (or Spring Boot), as well as with standards like OAuth2 or SAML.

What are Spring Security features?

Spring Security Features

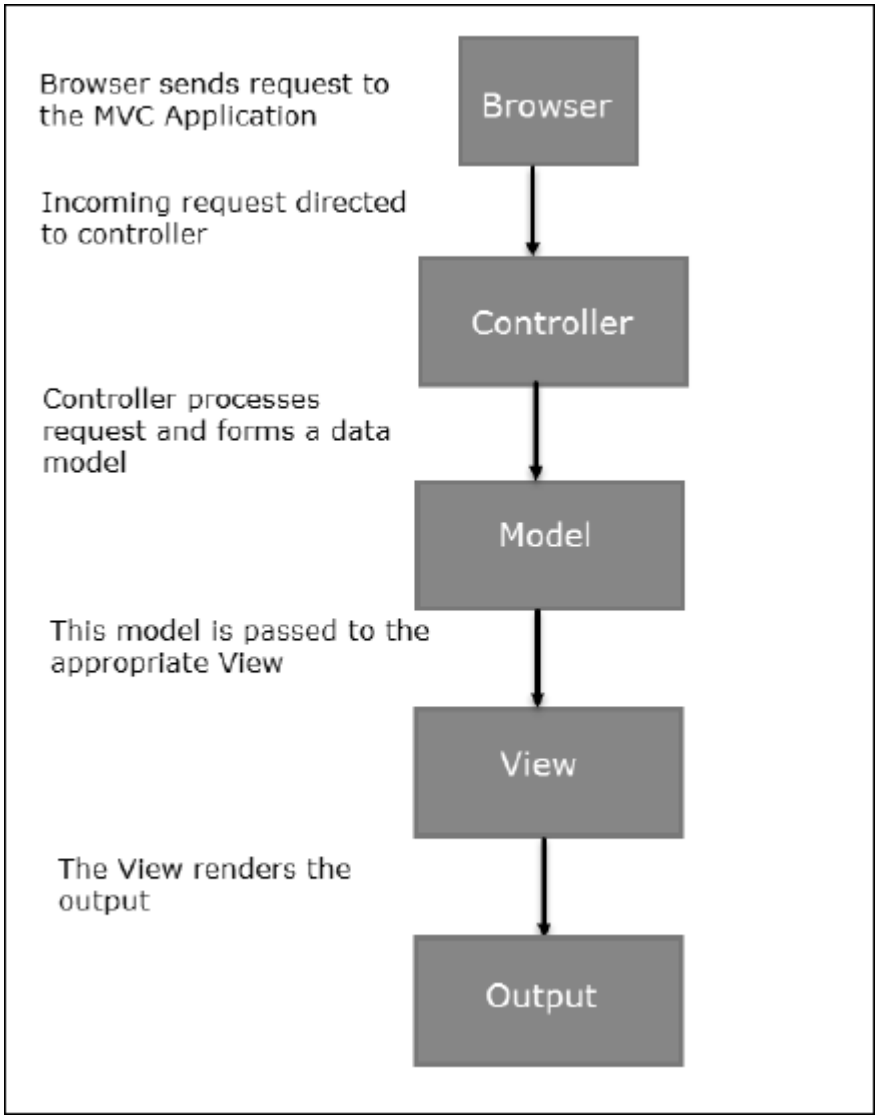
- LDAP (Lightweight Directory Access Protocol)
- Single sign-on.
- JAAS (Java Authentication and Authorization Service) LoginModule.
- Basic Access Authentication.
- Digest Access Authentication.
- Remember-me.
- Web Form Authentication.
- Authorization.

What is the advantage of Spring Security?

Advantages of Spring Security

Configuration **support to Java Programming Language**. Portable. Comprehensive support to tasks like authorization and authentication. Servlet API integration.

Flow of MVC?



Flow Steps

Step 1 − The client browser sends request to the MVC Application.

Step 2 − Global.ascx receives this request and performs routing based on the URL of the incoming request using the RouteTable, RouteData, UrlRoutingModule and MvcRouteHandler objects.

Step 3 − This routing operation calls the appropriate controller and executes it using the IControllerFactory object and MvcHandler object's Execute method.

Step 4 − The Controller processes the data using Model and invokes the appropriate method using ControllerActionInvoker object

Step 5 − The processed Model is then passed to the View, which in turn renders the final output.

A web service is a unit of managed code that can be remotely invoked using HTTP. That is, it can be activated using HTTP requests. Web services **allow you to expose the functionality of your existing code over the network**. Once it is exposed on the network, other applications can use the functionality of your program.

Web Services

- Benefits of Web Services
  - Interoperability in a heterogeneous environment
  - Business services through the Web
  - Integration with existing systems
  - Freedom of choice
  - Support more client types
  - Programming productivity
- Web services standards include:
  - Common markup language for communication: Web services use eXtensible Markup Language (**XML**) for the common markup language.
  - Common message format for exchanging information: Simple Object Access Protocol (**SOAP**) provides a common message format for Web services.
  - Common service specification formats: Web Services Description Language (**WSDL**) provides Web services with common specification formats.
  - Common means for service lookup: Universal Description, Discovery, and Integration (UDDI) specification defines a common means for looking up Web services.

Advantages of Web Services

- Because all communication is in XML…
  - Not limited to any single operating system
  - Not limited to any single programming language
- Cells can be developed in Microsoft .NET, Perl, Python, Java, etc.
  - Any language that supports REST or SOAP capability can be used
- Cells can exist on Windows, Linux, and Mac OS and communicate with each other
  - i.e. cells residing on a Windows platform can talk with those on a UNIX platform
- No restriction on how simple or complex a cell can be
  - XML tags the data
  - REST/SOAP transfers the data

Here are the advantages of utilizing web services are:

- Revealing the Existing Function on Framework. ...
- Interoperability. ...
- Ordered Protocol. ...
- Ease of Use. ...
- Re-Ease of Use. ...
- Send Capacity. ...
- Agility. ...
- Quality.

Interoperability - This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their solutions.

What is the main purpose of web service?

Web services are used for a variety of applications, but the most common is for **reusing code and connecting existing programs**. The web service method can help developers segment applications into components that can be used and reused for various needs.

What is web service definition?

A web service is **a software system that supports interoperable machine-to-machine interaction over a network**. It has an interface described in a machine-processable format (specifically, web Service Definition Language, or WSDL). web services fulfill a specific task or a set of tasks.

What are web services examples?

A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, **a client invokes a web service by sending an XML message, then waits for a corresponding XML response**.

How to convert jeson to java?

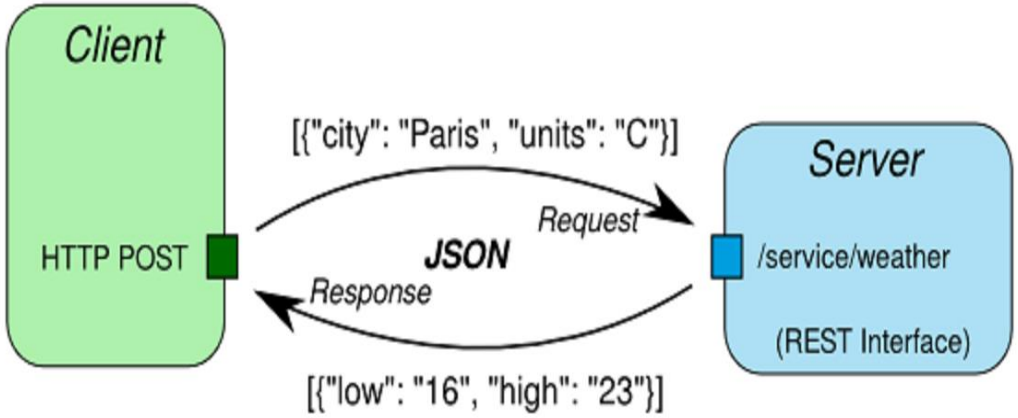
The **ObjectMapper** class is the most important class in the Jackson library. We can convert a JSON to Java Object using the **readValue()** method of **ObjectMapper** class, this method deserializes a JSON content from given JSON content String.

Syntax

```
public <T> readValue(String content, JavaType valueType) throws IOException, JsonParseException, JsonMappingException
```

Advantages of Spring MVC and its features

3 ways to convert JSON String to Java Object



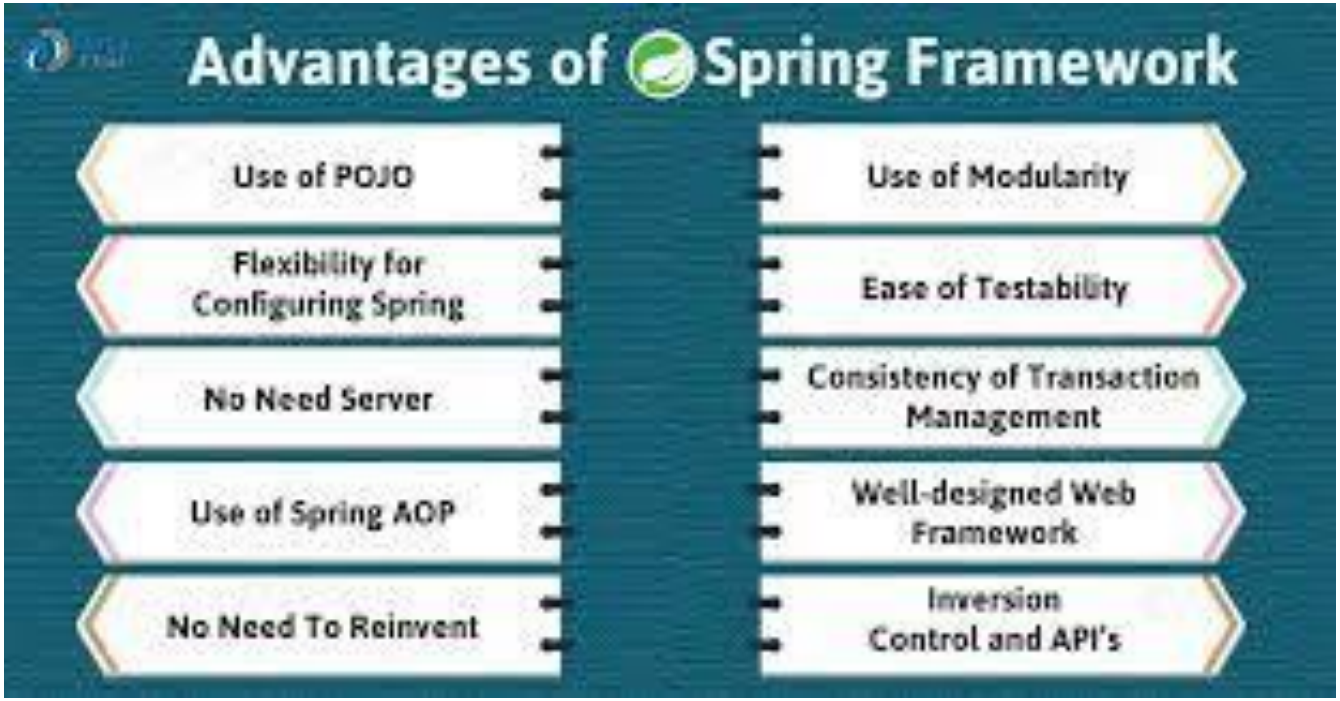
Advantages of Spring MVC and its features?

Advantages of Spring MVC Framework

Rapid development - The Spring MVC facilitates fast and parallel development. Reusable business code - Instead of creating new objects, it allows us to use the existing business objects. Easy to test - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.

Let's see some of the advantages of Spring MVC Framework:-

- **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
- **Light-weight** - It uses light-weight servlet container to develop and deploy your application.

- **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
- **Rapid development** - The Spring MVC facilitates fast and parallel development.
- **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
- **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
- **Flexible Mapping** - It provides the specific annotations that easily redirect the page.
- 

The main difference between these annotations is that **@ComponentScan scans for Spring components while @EnableAutoConfiguration is used for auto-configuring beans present in the classpath in Spring Boot applications**.

what is actuator in spring boot?

Actuator is **mainly used to expose operational information about the running application** — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it. Once this dependency is on the classpath, several endpoints are available for us out of the box.

In essence, Actuator brings production-ready features to our application.

Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency.

The main benefit of this library is that we can get production-grade tools without having to actually implement these features ourselves.

Actuator is mainly used to **expose operational information about the running application** — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

Once this dependency is on the classpath, several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways.

How to validate Spring MVC?

Spring MVC Validation Example

1. Add dependencies to pom.xml file. pom.xml. ...
2. Create the bean class. Employee.java. ...
3. Create the controller class. In controller class: ...
4. Provide the entry of controller in the web. xml file. ...
5. Define the bean in the xml file. spring-servlet.xml. ...
6. Create the requested page. ...
7. Create the other view components

Profiles in spring boot?

1. Overview

In this tutorial, we'll focus on introducing Profiles in Spring.

Profiles are a core feature of the framework — **allowing us to map our beans to different profiles** — for example, *dev*, *test*, and *prod*.

We can then activate different profiles in different environments to bootstrap only the beans we need.

Further reading:

Configuring Separate Spring DataSource for Tests

A quick, practical tutorial on how to configure a separate data source for testing in a Spring application.

Read more →

Properties with Spring and Spring Boot

Tutorial for how to work with properties files and property values in Spring.

Read more →

2. Use *@Profile* on a Bean

Let's start simple and look at how we can make a bean belong to a particular profile. **We use the *@Profile* annotation — we are mapping the bean to that particular profile**; the annotation simply takes the names of one (or multiple) profiles.

Consider a basic scenario: We have a bean that should only be active during development but not deployed in production.

We annotate that bean with a *dev* profile, and it will only be present in the container during development. In production, the *dev* simply won't be active:

```
@Component
@Profile("dev")
public class DevDatasourceConfig
```

As a quick sidenote, profile names can also be prefixed with a NOT operator, e.g., *!dev*, to exclude them from a profile.

In the example, the component is activated only if *dev* profile is not active:

```
@Component
@Profile("!dev")
public class DevDatasourceConfig
```

3. Declare Profiles in XML

Profiles can also be configured in XML. The *<beans>* tag has a *profile* attribute, which takes comma-separated values of the applicable profiles:

```
<beans profile="dev">
   <bean id="devDatasourceConfig"
    class="org.baeldung.profiles.DevDatasourceConfig" />
</beans>
```

4. Set Profiles

The next step is to activate and set the profiles so that the respective beans are registered in the container.

This can be done in a variety of ways, which we'll explore in the following sections.

4.1. Programmatically via *WebApplicationInitializer* Interface

In web applications, *WebApplicationInitializer* can be used to configure the *ServletContext* programmatically.

It's also a very handy location to set our active profiles programmatically:

```
@Configuration
public class MyWebApplicationInitializer
 implements WebApplicationInitializer {

   @Override
   public void onStartup(ServletContext servletContext) throws ServletException {
```

```
    servletContext.setInitParameter(
      "spring.profiles.active", "dev");
  }
}
```

4.2. Programmatically via *ConfigurableEnvironment*

We can also set profiles directly on the environment:

```
@Autowired
private ConfigurableEnvironment env;
...
env.setActiveProfiles("someProfile");
```

4.3. Context Parameter in *web.xml*

Similarly, we can define the active profiles in the *web.xml* file of the web application, using a context parameter:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/app-config.xml</param-value>
</context-param>
<context-param>
  <param-name>spring.profiles.active</param-name>
  <param-value>dev</param-value>
</context-param>
```

4.4. JVM System Parameter

The profile names can also be passed in via a JVM system parameter. These profiles will be activated during application startup:

```
-Dspring.profiles.active=dev
```

4.5. Environment Variable

In a Unix environment, **profiles can also be activated via the environment variable**:

```
export spring_profiles_active=dev
```

4.6. Maven Profile

Spring profiles can also be activated via Maven profiles, by **specifying the *spring.profiles.active* configuration property**.

In every Maven profile, we can set a *spring.profiles.active* property:

```
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <spring.profiles.active>dev</spring.profiles.active>
    </properties>
  </profile>
  <profile>
    <id>prod</id>
    <properties>
      <spring.profiles.active>prod</spring.profiles.active>
    </properties>
  </profile>
</profiles>
```

Its value will be used to replace the *@spring.profiles.active@* placeholder in *application.properties*:

```
spring.profiles.active=@spring.profiles.active@
```

Now we need to enable resource filtering in *pom.xml*:

```
<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
```

```
    </resource>
  </resources>
  ...
</build>
```

and append a *-P* parameter to switch which Maven profile will be applied:

```
mvn clean package -Pprod
```

This command will package the application for *prod* profile. It also applies the *spring.profiles.active* value *prod* for this application when it is running.

4.7. *@ActiveProfile* in Tests

Tests make it very easy to specify what profiles are active using the *@ActiveProfile* annotation to enable specific profiles:

```
@ActiveProfiles("dev")
```

So far, we've looked at multiple ways of activating profiles. Let's now see which one has priority over the other and what happens if we use more than one, from highest to lowest priority:

1. Context parameter in *web.xml*
2. *WebApplicationInitializer*
3. JVM System parameter
4. Environment variable
5. Maven profile

5. The Default Profile

Any bean that does not specify a profile belongs to the *default* profile.

Spring also provides a way to set the default profile when no other profile is active — by using the *spring.profiles.default* property.

6. Get Active Profiles

Spring's active profiles drive the behavior of the *@Profile* annotation for enabling/disabling beans. However, we may also wish to access the list of active profiles programmatically.

We have two ways to do it, **using *Environment* or *spring.active.profile*.**

6.1. Using *Environment*

We can access the active profiles from the *Environment* object by injecting it:

```
public class ProfileManager {
    @Autowired
    private Environment environment;

    public void getActiveProfiles() {
        for (String profileName : environment.getActiveProfiles()) {
            System.out.println("Currently active profile - " + profileName);
        }
    }
}
```

6.2. Using *spring.active.profile*

Alternatively, we could access the profiles by injecting the property *spring.profiles.active*:

```
@Value("${spring.profiles.active}")
private String activeProfile;
```

Here, our *activeProfile* variable **will contain the name of the profile that is currently active**, and if there are several, it'll contain their names separated by a comma.

However, we should **consider what would happen if there is no active profile at all.** With our code above, the absence of an active profile would prevent the application context from being created. This would result in an *IllegalArgumentException* owing to the missing placeholder for injecting into the variable.

In order to avoid this, we can **define a default value**:

What is Java Executor Framework?

With the increase in the number of cores available in the processors nowadays, coupled with the ever-increasing need to achieve more throughput, multi-threading APIs are getting quite popular. Java provides its own multi-threading framework called the Java Executor Framework.

Java executor framework (java.util.concurrent.Executor), released with the JDK 5 is used to run the Runnable objects without creating new threads every time and mostly re-using the already created threads. We all know that there are two ways to create a thread in java. If you want to read more about their comparison, read how to create threads in Java.
The java.util.concurrent.Executors provide factory methods that are being used to create ThreadPools of worker threads. Thread pools overcome this issue by keeping the threads alive and reusing the threads. Any excess tasks flowing in that the threads in the pool can handle are held in a Queue. Once any of the threads get free, they pick up the next task from this queue. This task queue is essentially unbounded for the out-of-box executors provided by the JDK.
Some types of Java Executors are listed below:

1. SingleThreadExecutor
2. FixedThreadPool(n)+
3. CachedThreadPool
4. ScheduledExecutor

Let us discuss these popular java executors to some details what exactly they do to get a better idea prior to implementing the same.

Executor 1: SingleThreadExecutor
A single thread pool can be obtained by calling the static *newSingleThreadExecutor()* method of the Executors class. It is used to execute tasks sequentially.
Syntax:

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

Executor 2: FixedThreadPool(n)
As the name indicates, it is a thread pool of a fixed number of threads. The tasks submitted to the executor are executed by the n threads and if there is more task they are stored on a LinkedBlockingQueue. It uses Blocking Queue.

Syntax:

```
ExecutorService fixedPool = Executors.newFixedThreadPool(2);
```

Executor 3: CachedThreadPool

Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available. Calls to execute will reuse previously constructed threads if available. If no existing thread is available, a new thread will be created and added to the pool. It uses a SynchronousQueue queue.

```
ExecutorService executorService = Executors.newCachedThreadPool();
```

Executor 4: ScheduledExecutor

Scheduled executors are based on the interface ScheduledExecutorService which extends the ExecutorService interface. This executor is used when we have a task that needs to be run at regular intervals or if we wish to delay a certain task.

```
ScheduledExecutorService scheduledExecService = Executors.newScheduledThreadPool(1);
```

- The tasks can be scheduled using either of the two methods:
  - **scheduleAtFixedRate**: Executes the task with a fixed interval, irrespective of when the previous task ended.
  - **scheduleWithFixedDelay**: This will start the delay countdown only after the current task completes.`

Syntax:

```
scheduledExecService.scheduleAtFixedRate(Runnable command, long initialDelay, long period, TimeUnit unit)

scheduledExecService.scheduleWithFixedDelay(Runnable command, long initialDelay, long period, TimeUnit unit)
```

Future Object

The result of the task submitted for execution to an executor can be accessed using the java.util.concurrent.The future object returned by the executor. Future can be thought of as a promise made to the caller by the executor. The future interface is mainly used to get the results of Callable results. whenever the task execution is completed, it is set in this Future object by the executor.

Syntax:

```
Future<String> result = executorService.submit(callableTask);
```

Implementation: Creating and Executing a Simple Executor in which we will create a task and execute it in a fixed pool

- The Task class implements Callable and is parameterized to String type. It is also declared to throw Exception.
- Now in order to execute task in class “Task” we have to instantiate the Task class and are passing it to the executor for execution.
- Print and display the result that is returned by the Future object

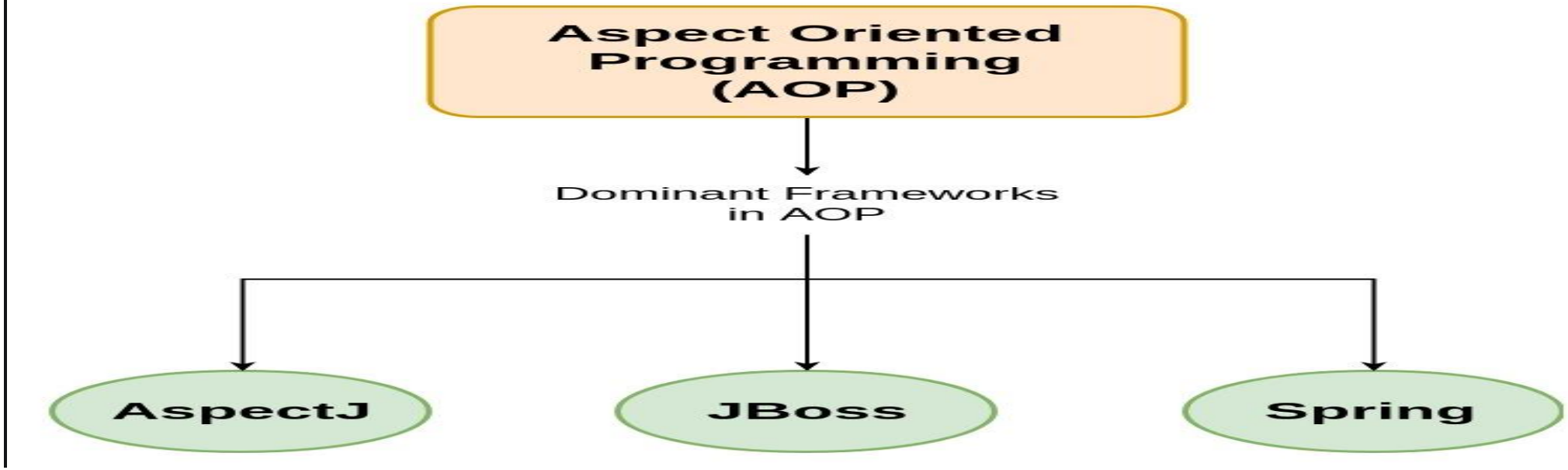
Aspect Oriented Programming and AOP in Spring Framework

- Last Updated : 09 Feb, 2022

Aspect oriented programming(AOP) as the name suggests uses aspects in programming. It can be defined as the breaking of code into different modules, also known as modularisation, where the aspect is the key unit of modularity. Aspects enable the implementation of crosscutting concerns such as- transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behaviour that is the advice to the existing code. For example- Security is a crosscutting concern, in many methods in an application security rules can be applied, therefore repeating the code at every method, define the functionality in a common class and control were to apply that functionality in the whole application.

Dominant Frameworks in AOP:

AOP includes programming methods and frameworks on which modularisation of code is supported and implemented. Let's have a look at the three **dominant frameworks in AOP**:



Diff bet session factory and application context?

The ApplicationContext comes with advanced features, including several that are geared towards enterprise applications, while the BeanFactory comes with only basic features. Therefore, it's generally recommended to use the ApplicationContext, and we should use BeanFactory only when memory consumption is critical.

BeanFactory	ApplicationContext
It is an interface defined in org.springframework.beans.factory.**BeanFactory**	It is an interface defined in org.springframework.context.**ApplicationContext**
It uses Lazy initialization	It uses Eager/ Aggressive initialization
It explicitly provides a resource object using the syntax	It creates and manages resource objects on its own
It doesn't supports internationalization	It supports internationalization
It doesn't supports annotation based dependency	It supports annotation based dependency

1. What Is a Circular Dependency?

A circular dependency occurs when a bean A depends on another bean B, and the bean B depends on bean A as well:

Bean A → Bean B → Bean A

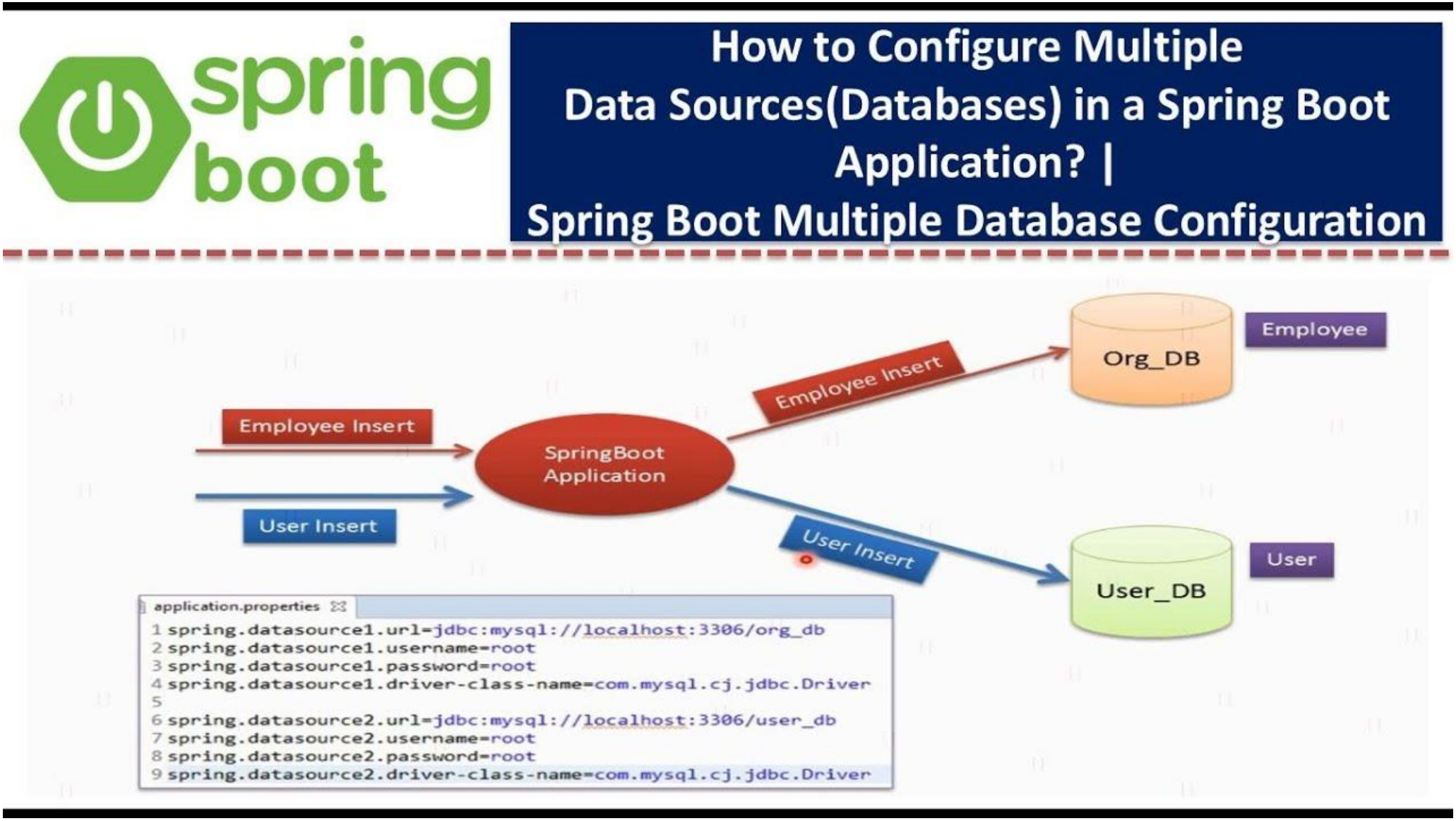
Of course, we could have more beans implied:

Bean A → Bean B → Bean C → Bean D → Bean E → Bean A

Multiple Data Sources with Spring Boot

1. Maven Setup. To set up our Spring Boot project, we need to add spring-boot-starter-data-jpa dependency to the pom. ...
2. DataSource Configurations. ...
3. JPA Entities. ...
4. Package Structure. ...
5. JPA Repositories. ...

6. Spring Configuration Classes. ...
7. Testing.



1. Introduction

In this tutorial, we're going to show how to **inject *Java* collections using the Spring framework**.

Simply put, we'll demonstrate examples with the *List*, *Map*, *Set* collection interfaces.

2. *List* With *@Autowired*

Let's create an example bean:

```
public class CollectionsBean {

    @Autowired
    private List<String> nameList;

    public void printNameList() {
        System.out.println(nameList);
    }
}
```

Here, we declared the *nameList* property to hold a *List* of *String* values.

In this example, we use field injection for *nameList*. Therefore, we put the *@Autowired* annotation.

To learn more about the dependency injection or different ways to implement it, check out this guide.

After, we register the *CollectionsBean* in the configuration setup class:

```
@Configuration
public class CollectionConfig {

    @Bean
    public CollectionsBean getCollectionsBean() {
        return new CollectionsBean();
```

```
}

@Bean
public List<String> nameList() {
   return Arrays.asList("John", "Adam", "Harry");
}
```

Spring - Injecting Collection

Previous Page

Next Page

You have seen how to configure primitive data type using **value** attribute and object references using **ref** attribute of the <property> tag in your Bean configuration file. Both the cases deal with passing singular value to a bean.

Now what if you want to pass plural values like Java Collection types such as List, Set, Map, and Properties. To handle the situation, Spring offers four types of collection configuration elements which are as follows −

Sr.No	Element & Description
1	**<list>** This helps in wiring ie injecting a list of values, allowing duplicates.
2	**<set>** This helps in wiring a set of values but without any duplicates.
3	**<map>** This can be used to inject a collection of name-value pairs where name and value can be of any type.

4	**<props>** This can be used to inject a collection of name-value pairs where the name and value are both Strings.

You can use either <list> or <set> to wire any implementation of java.util.Collection or an **array**.

You will come across two situations (a) Passing direct values of the collection and (b) Passing a reference of a bean as one of the collection elements.

how to import one xml file into another file?

Linked

1. Import separate XAML as ResourceDictionary and assign x:Key.
2. Extend logback configuration.
3. Reading and validating xml document with external entities using Qt.

what are the diff ways of creating autowired class?

Spring Autowiring modes

No	⇒	No autowiring at all. Bean Reference must be defined via a ref element.
byName	⇒	It will look for a bean named exactly the same as the property which needs to be autowired.
byType	⇒	It will check the data type of the bean is same as data type of other bean property.
Constructor	⇒	This is analogous to byType, but applies to constructor arguments.
autodetect	⇒	Chooses constructor or byType through Introspection of the bean class.

how to input from user in IOC?

IoC Container

1. IoC Container
2. Using BeanFactory
3. Using ApplicationContext

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. The main tasks performed by IoC container are:

- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

in how many ways can handle exception in Spring?

Spring Boot - Exception Handling

Previous Page

Next Page

Handling exceptions and errors in APIs and sending the proper response to the client is good for enterprise applications. In this chapter, we will learn how to handle exceptions in Spring Boot.

Before proceeding with exception handling, let us gain an understanding on the following annotations.

Controller Advice

The @ControllerAdvice is an annotation, to handle the exceptions globally.

Exception Handler

The @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

You can use the following code to create @ControllerAdvice class to handle the exceptions globally −

```
package com.tutorialspoint.demo.exception;

import org.springframework.web.bind.annotation.ControllerAdvice;

@ControllerAdvice
  public class ProductExceptionController {
}
```

Define a class that extends the RuntimeException class.

```
package com.tutorialspoint.demo.exception;

public class ProductNotfoundException extends RuntimeException {
  private static final long serialVersionUID = 1L;
}
```

You can define the @ExceptionHandler method to handle the exceptions as shown. This method should be used for writing the Controller Advice class file.

```
@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {
}
```

Now, use the code given below to throw the exception from the API.

```
@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
public ResponseEntity<Object> updateProduct() {
  throw new ProductNotfoundException();
}
```

The complete code to handle the exception is given below. In this example, we used the PUT API to update the product. Here, while updating the product, if the product is not found, then return the response error message as “Product not found”. Note that the **ProductNotFoundException** exception class should extend the **RuntimeException**.

inner bean and outer bean scope?

Inner beans are always anonymous and they are always created with the outer bean. It is not possible to inject inner beans into collaborating beans other than into the enclosing bean. So **an inner bean has no scope** and basically can't be used by anything other than the enclosing bean.

Spring - Bean Scopes

Previous Page

Next Page

When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be **prototype**. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be **singleton**.

The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

Sr.No.	Scope & Description
1	**singleton** This scopes the bean definition to a single instance per Spring IoC container (default).
2	**prototype** This scopes a single bean definition to have any number of object instances.
3	**request**

	This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4	**session** This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5	**global-session** This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

diff bet singleton and mvc design pattern?

Singleton design pattern comes under the category of creational design pattern. In this only one instance of the class is created which is used by all the callers. Observer and MVC come under the category of behavioural design pattern. These concentrate on the interaction between objects.

Singleton, Observer and MVC are three different design patterns which have been designed to model business applications. They fall under three different categories.

Singleton design pattern comes under the category of creational design pattern. In this only one instance of the class is created which is used by all the callers.

Observer and MVC come under the category of behavioural design pattern. These concentrate on the interaction between objects.

In Observer pattern one or many objects may be dependent on another object such that when the main object is modified the waiting/observing objects also need to be notified and undergo some operation.

In MVC design pattern there is a clear separation of the business logic, presentation logic and data. In this there is a front end View(presentation layer) which is viewed by the user and then the back-end Model(data) layer which contains the related values required. The middle tier of Controller(business logic) regulates the flow of data between the model and the view.

how to run spring boot application?

How to Run Spring Boot Application

1. Step 1: Open the Spring Initializr https://start.spring.io/.
2. Step 2: Select the Spring Boot version 2.2. ...
3. Step 3: Provide the Group name. ...
4. Step 4: Provide the Artifact. ...
5. Step 5: Add the Spring Web dependency.
6. Step 6: Click on the Generate button. ...
7. Step 7: Extract the jar file.

by using which method in springboot we can sent request to the front end?

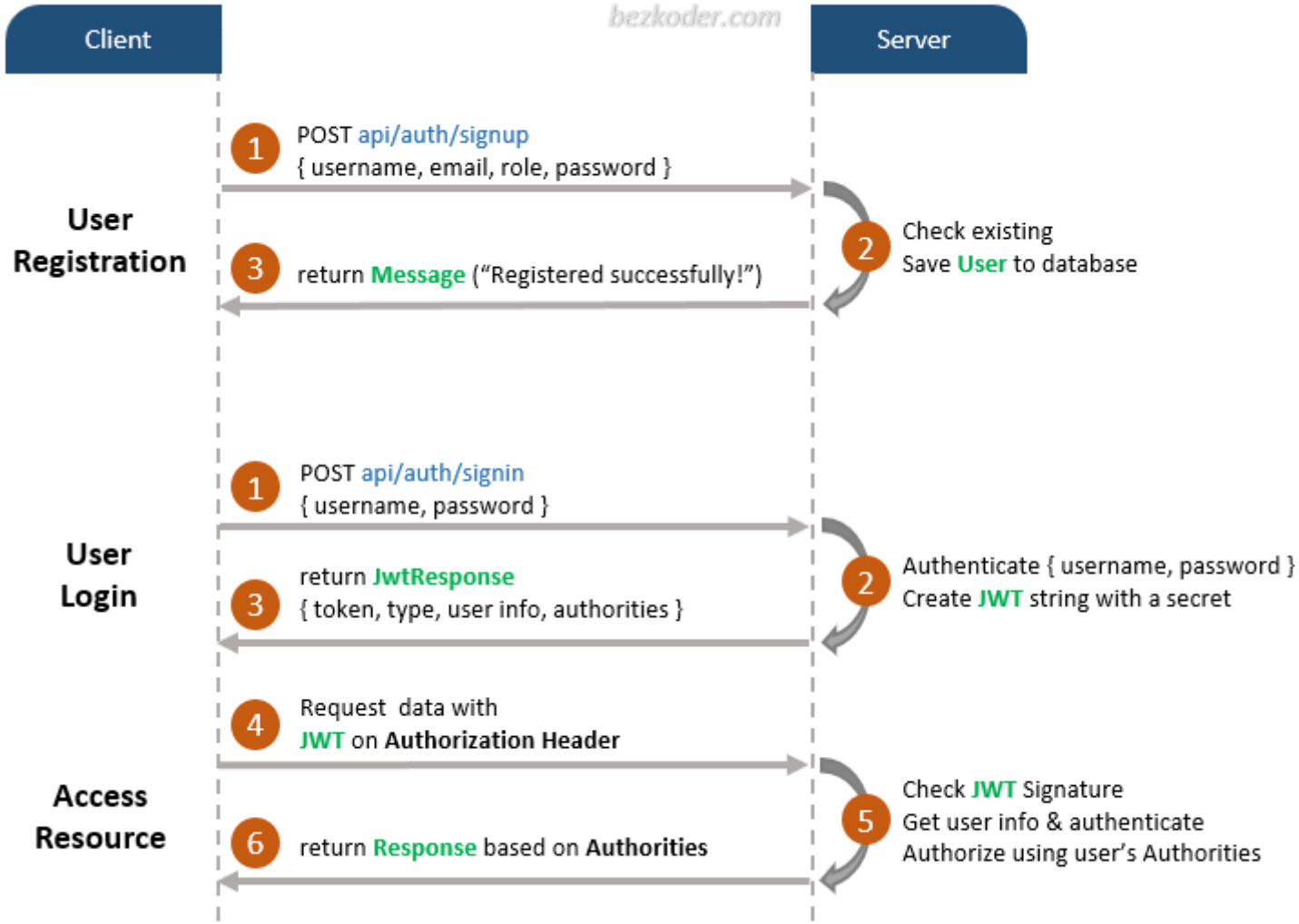
I am Planning to build a web application using Spring Boot as restful service. my spring boot web restful application should be accessible by other application as well. In case if any one accessing the rest service from other application then my application should work as expected.

```
@Controller
public class GreetingController {
   @RequestMapping("/greeting")
   public String greeting(@RequestParam(value="name", required=false,  defaultValue="World") String name, Model model) {
      model.addAttribute("name", name);
```

How to authenticate in spring boot?

1. Start with Spring Boot and Thymeleaf.
2. Start Your Spring Boot Application.
3. Configure User Authentication in Your Spring Boot App with OAuth 2.0.
4. Add User Authentication via OAuth 2.0 to the Spring Boot Project.
5. Start Your Spring Boot App with OAuth 2.0 SSO.
6. Create the Restricted Controller Method and Thymeleaf Template.

What are mapping used in jpa?



What are mapping used in jpa?

Association mappings are one of the key features of JPA and Hibernate. They **model the relationship between two database tables as attributes in your domain model**. That allows you to easily navigate the associations in your domain model and JPQL or Criteria queries.

Why we used 400 error code?

A 400 Bad Request Error **occurs when a request sent to the website server is incorrect or corrupt, and the server receiving the request can't understand it**. Occasionally, the problem is on the website itself, and there's not much you can do about that.

Why we used apache tomcat server in our application?

It is mainly used **to provide the foundation for hosting Java servlets**. The Apache Tomcat works in the center while Java Server Pages and Servlet produce the dynamic pages. It is one of the server-side programming languages that facilitate the developer to run and perform independent dynamic content creation.

What is application server for deployment?

An application server is **a server that hosts applications**. Application server frameworks are software frameworks for building application servers. An application server framework provides both facilities to create web applications and a server environment to run them.

How will get the record in rest api?

You **use the sObject Rows resource to retrieve field values from a record**. Specify the fields you want to retrieve in the fields parameter and use the GET method of the resource. You can use the GET method of the sObject Rows by External ID resource to retrieve records with a specific external ID

Return Value of put() in HashMap

1. if abc key already exists with value OFFLINE , ret is equal to OFFLINE .

2. if abc key already exists with value ONLINE , ret is equal to ONLINE .
3. if abc key did not exist, then ret is equal to null .

Containers are **the interface between a component and the low-level, platform-specific functionality that supports the component**. Before it can be executed, a web, enterprise bean, or application client component must be assembled into a Java EE module and deployed into its container.

Applet container

The Applet container manages Java applets. An Applet is a Java program that can be embedded into a web page. Most web pages are rendered using HTML/XMLbased technology. By embedding the tags <APPLET> and </APPLET> a browser will load a Java applet, which can use the Java AWT/Swing interface APIs, allowing a traditional client-like application to run within the browser. The Applet container manages the execution of the applet, and contains the web browser.

Web container

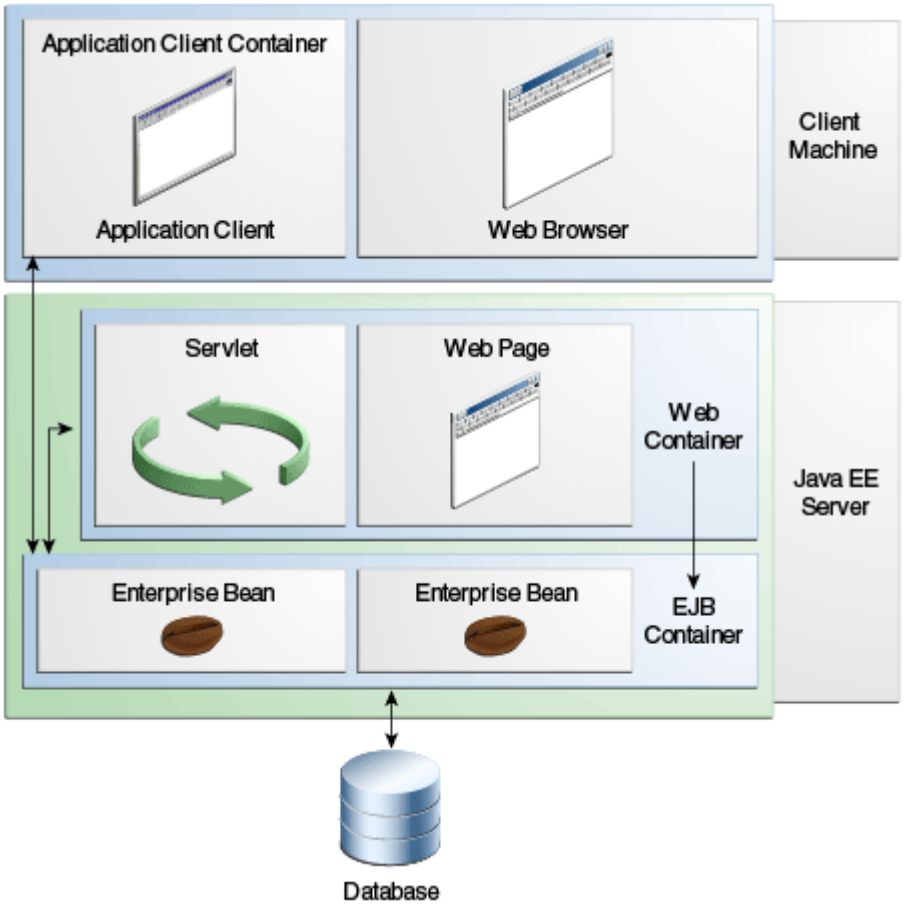
The Web container, also known as a Servlet container, provides web-related services. In a nutshell, this is the component of a web-server which serves web content, web-services, facilitates web-security, application deployment, and other key services. The following diagram shows the availability of the Java EE 6 APIs in the web container:

EJB container

The EJB (Enterprise JavaBean) container manages the services of the EJB API and provides an environment for running the enterprise components of a JEE application. Enterprise JavaBeans are used in distributed applications, and facilitate transaction services and appropriate low-level implementations of transaction management and coordination, as required by key business processes. They are essentially the business components of an application.
The EJB container also manages database connections and pooling, threads, and sockets on behalf of enterprise beans, as well as state and session management. The following diagram shows the availability of the Java EE 6 APIs in the EJB container:

Application client container

An application client runs on a user's client machine and provides a traditional rich Graphical User Interface (GUI) created from the Swing or the Abstract Window Toolkit (AWT) API. Application client's access enterprise beans running in the business tier—which we explained earlier—run in the EJB container. An application client can use RMI (Remote Method Invocation) or other protocols, such as SOAP (Simple Object Access Protocol), over HTTP (Hypertext Transfer Protocol). The following diagram shows the Java EE 6 APIs within the application client container:



What is immutable class in Java?

Immutable class in java means that **once an object is created, we cannot change its content**. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable.

STEPS TO CREATE AN IMMUTABLE CLASS

- DECLARE THE CLASS AS FINAL SO IT CAN'T BE EXTENDED.
- MAKE ALL FIELDS PRIVATE SO THAT DIRECT ACCESS IS NOT ALLOWED.
- DON'T PROVIDE SETTER METHODS FOR VARIABLES.
- MAKE ALL MUTABLE FIELDS FINAL SO THAT ITS VALUE CAN BE ASSIGNED ONLY ONCE.
- INITIALIZE ALL THE FIELDS VIA A CONSTRUCTOR PERFORMING DEEP COPY.
- PERFORM CLONING OF OBJECTS IN THE GETTER METHODS TO RETURN A COPY RATHER THAN RETURNING THE ACTUAL OBJECT REFERENCE.

G_E_E_K_Y_G_I_R_L

Singleton class?

Singleton Pattern says that just**"define a class that has only one instance and provides a global point of access to it".**

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

There are two forms of singleton design pattern

- **Early Instantiation:** creation of instance at load time.
- **Lazy Instantiation:** creation of instance when required.

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.

Usage of Singleton design pattern

- Singleton pattern is mostly used in multi-threaded and database applications. It is used in logging, caching, thread pools, configuration settings etc.

Constructor Chaining

In constructor chain, a constructor is called from another constructor in the same class this process is known as **constructor chaining.** It occurs through inheritance. When we create an instance of a derived class, all the constructors of the inherited class (base class) are first invoked, after that the constructor of the calling class (derived class) is invoked.

We can achieve constructor chaining in two ways:

- **Within the same class:** If the constructors belong to the same class, we use **this**

- **From the base class:** If the constructor belongs to different classes (parent and child classes), we use the **super** keyword to call the constructor from the base class.

Method Chaining in Java

In Java, **method chaining** is the chain of methods being called one after another. It is the same as constructor chaining but the only difference is of method and constructor. In this section, we will discuss the **method chaining in Java.**

Method Chaining

Method chaining is a common syntax to invoke multiple methods calls in OOPs. Each method in chaining returns an object. It violates the need of intermediate variables.

In other words, the method chaining can be defined as if we have an object and we call methods on that object one after another is called method chaining. For example,

```
1. obj.method1().method2().method3();
```

In the above statement, we have an object (obj) and calling method1() then method2(), after that the method3(). So, calling or invoking methods one after another is known as method chaining.

It is also known as **parameter idiom** or **named parameter idiom**. Sometimes it is also known as a train wreck because of the increase in the number of methods even though line breaks are often added between methods.

Applications of Method Chaining

- It is used to implement **method cascading**.
- It is also used to implement in **fluent interfaces**.

Copy Constructor

In Java, a copy constructor is a special type of constructor that creates an object using another object of the same Java class. It returns a duplicate copy of an existing object of the class.

We can assign a value to the final field but the same cannot be done while using the clone() method. It is used if we want to create a deep copy of an existing object. It is easier to implement in comparison to the clone() method.

Use of Copy Constructor

We can use the copy constructor if we want to:

- Create a copy of an object that has multiple fields.
- Generate a deep copy of the heavy objects.
- Avoid the use of the Object.clone() method.

Advantages of Copy Constructor

- If a field declared as final, the copy constructor can change it.
- There is no need for typecasting.
- Its use is easier if an object has several fields.
- Addition of field to the class is easy because of it. We need to change only in the copy constructor.

Creating a Copy Constructor

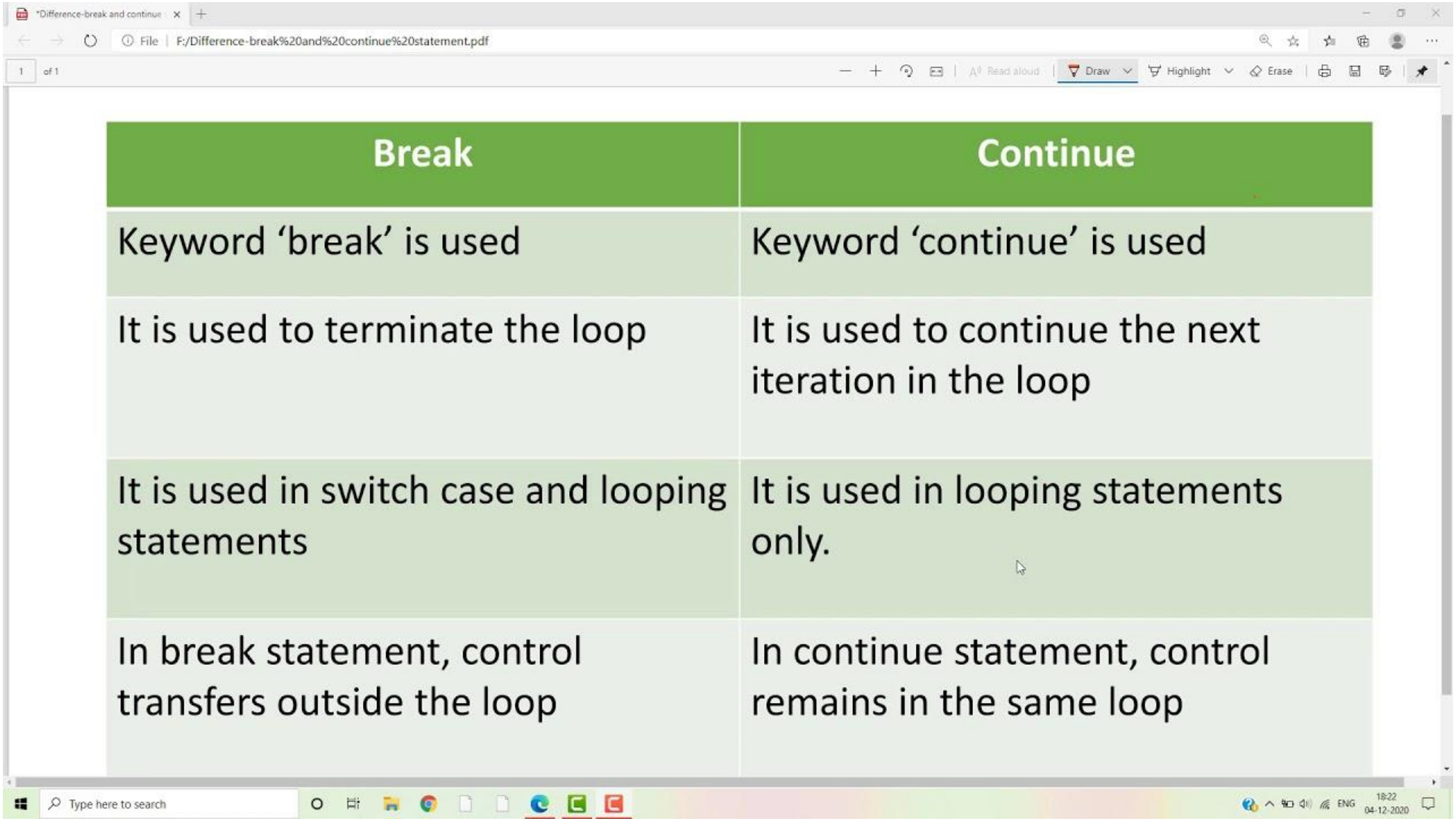
To create a copy constructor in Java, follow the steps given below:

- Create a constructor that accepts an object of the same class as a parameter.

Stack	Heap
Memory is managed for you	Memory management needs to be done manually
Smaller in Size	Larger in Size
Access is easier and faster and cache friendly	Causes more cache misses because of being dispersed throughout the memory
Not flexible, allotted memory cannot be changed	Flexible and allotted memory can be altered
Faster access, allocation and deallocation	Slower access, allocation and deallocation
Elements' scope is limited to their threads	Elements are globally accessible in the application
OS allocates the stack when the thread is created	OS is called by the language in the runtime to allocate the heap for the application

difference between constructor and method.?

- **Constructor is used to initialize an object whereas method is used to exhibits functionality of an object.**
- **Constructors are invoked implicitly whereas methods are invoked explicitly.**
- **Constructor does not return any value where the method may/may not return a value.**
- **In case constructor is not present, a default constructor is provided by java compiler. In the case of a method, no default method is provided.**
- **Constructor should be of the same name as that of class. Method name should not be of the same name as that of class.**



Break	Continue
Keyword ‘break’ is used	Keyword ‘continue’ is used
It is used to terminate the loop	It is used to continue the next iteration in the loop
It is used in switch case and looping statements	It is used in looping statements only.
In break statement, control transfers outside the loop	In continue statement, control remains in the same loop

Staic Variable	Non Static Variable
Staic keyword must be used to declare	non used static keyword to declare variable. Also called as Object variable.
Called by classname.variable	Called with Ojbect.variable
Only one copy would be generated.	Multiple copies could be generated along with creation of object.
Initialization done before object creation.	Initialization done after object creation.

Static Method	Non-Static Method
1. *The static* keyword must be used before the method name	1. No need to use the *static* keyword before the method name
2. It is called using the class (className.methodName)	2. It is can be called like any general method
3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class

Java ClassLoader

Java ClassLoader is an abstract class. It belongs to a **java.lang** package. It loads classes from different resources. Java ClassLoader is used to load the classes at run time. In other words, JVM performs the linking process at runtime. Classes are loaded into the JVM according to need. If a loaded class depends on another class, that class is loaded as well. When we request to load a class, it delegates the class to its parent. In this way, uniqueness is maintained in the runtime environment. It is essential to execute a Java program.

Types of ClassLoader

In Java, every ClassLoader has a predefined location from where they load class files. There are following types of ClassLoader in Java:

Bootstrap Class Loader: It loads standard JDK class files from rt.jar and other core classes. It is a parent of all class loaders. It doesn't have any parent. When we call String.class.getClassLoader() it returns null, and any code based on it throws NullPointerException. It is also called Primordial ClassLoader. It loads class files from jre/lib/rt.jar. For example, java.lang package class.

Extensions Class Loader: It delegates class loading request to its parent. If the loading of a class is unsuccessful, it loads classes from jre/lib/ext directory or any other directory as java.ext.dirs. It is implemented by sun.misc.Launcher$ExtClassLoader in JVM.

System Class Loader: It loads application specific classes from the CLASSPATH environment variable. It can be set while invoking program using -cp or classpath command line options. It is a child of Extension ClassLoader. It is implemented by sun.misc.Launcher$AppClassLoader class. All Java ClassLoader implements java.lang.ClassLoader.

how does the constructor work?

A constructor in Java is **a special method that is used to initialize objects**. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. In Java, a constructor is a block of codes similar to the method.

what is design pattern and explain?

Design patterns represent the best practices used by experienced object-oriented software developers. Design patterns are solutions to general problems that software developers faced during software development. These solutions were obtained by trial and error by numerous software developers over quite a substantial period of time.

This tutorial will take you through step by step approach and examples using Java while learning Design Pattern concepts.

what is static and its uses?

The static keyword in Java is **mainly used for memory management**. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class.

What is Garbage Collection?

- ***Garbage collection*** is the process which reclaims the memory allocated by objects which are unreferenced .
- Since objects are stored in the heap memory it can be considered as a mechanism to reclaim the heap memory
- ***Garbage collector*** is a JVM daemon thread which perform the task of garbage collection
- An object is a candidate for garbage collection if and only if there is no reference to it.
- Any object becoming unreferenced does not mean that it will be immediately garbage collected , it only means that the object is candidate for getting collected in the next garbage collector cycle.
- JVM runs the garbage collector based on several predefined algorithms mostly when it is in need of memory

what is reflection API in java?

Java Reflection API

Java Reflection is a *process of examining or modifying the run time behavior of a class at run time*.

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

Where it is used

The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g., Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.

- **how can we make singleton pattern synchronized?**

To make a singleton class thread-safe, **getInstance() method** is made synchronized so that multiple threads can't access it simultaneously.

...
Pros:

- Object is created only if it is needed. ...
- Exception handling is also possible in method.
- Every time a condition of null has to be checked.

.

What is factory design pattern?

The Factory Design Pattern or Factory Method Design Pattern is **one of the most used design patterns in Java**. According to GoF, this pattern “defines an interface for creating an object, but let subclasses decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses”.

Final

- Keyword used in context with class, method and reference /primitive type variable declaration
- Class-Cannot take part in inheritance
- Method-cannot overridden
- Primitive type variable declared as final is a constant

Finally

- Keyword used in context with exception handling
- Block-will execute any how irrespective of any thrown exception
- Can be followed by try-catch/without it

Finalize

- Method

  A protected method of java.lang.object
- This method is called by java garbage collected which is least recently used.
- Can be called Explicitly from an application program.

Difference between ArrayList & Vector

ArrayList	Vector
Every method present ArrayList is non synchronize	Every method present in LinkedList is synchronize
At a time multiple threads are allowed to operate on ArrayList Object and hence **ArrayList** is not thread safe	At a time only one thread is allowed to operate on **Vector** Object is thread safe
Threads are not required to wait to operate on ArrayList, hence relatively performance is high.	Threads are required to wait to operate on Vector Object and hence relatively performance is low
Introduced in 1.2 version And it is non legacy class	Introduced in 1.0 version and it is a legacy class

what is Map ?

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

scenario to pick linklist over arraylist?

Difference between ArrayList and LinkedList in Java

ArrayList	LinkedList
ArrayList internally uses a dynamic array to store the elements	LinkedList internally uses a doubly linked list to store the elements
Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
ArrayList consumes less memory than LinkedList	A LinkedList consumes more memory than an ArrayList because it also stores the next and previous references along with the data.
An ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
ArrayList is better for storing and accessing data.	LinkedList is **better for manipulating** data.

LinkedList should be used **where modifications to a collection are frequent like addition/deletion operations**. LinkedList is much faster as compare to ArrayList in such cases. In case of read-only collections or collections which are rarely modified, ArrayList is suitable.

- **copy on arraylist?**
    - CopyOnWriteArrayList is a thread-safe variant of ArrayList where operations which can change the ArrayList (add, update, set methods) creates a clone of the underlying array.
    - CopyOnWriteArrayList is to be used in a Thread based environment where read operations are very frequent and update operations are rare.
    - Iterator of CopyOnWriteArrayList will never throw ConcurrentModificationException.
    - Any type of modification to CopyOnWriteArrayList will not reflect during iteration since the iterator was created.
    - List modification methods like remove, set and add are not supported in the iteration. This method will throw UnsupportedOperationException.

What is Hashing

It is the process of converting an object into an integer value. The integer value helps in indexing and faster searches.

What is HashMap

HashMap is a part of the Java collection framework. It uses a technique called Hashing. It implements the map interface. It stores the data in the pair of Key and Value. HashMap contains an array of the nodes, and the node is represented as a class. It uses an array and LinkedList data structure internally for storing Key and Value. There are four fields in HashMap.

Before understanding the internal working of HashMap, you must be aware of hashCode() and equals() method.

- **equals():** It checks the equality of two objects. It compares the Key, whether they are equal or not. It is a method of the Object class. It can be overridden. If you override the equals() method, then it is mandatory to override the hashCode() method.
- **hashCode():** This is the method of the object class. It returns the memory reference of the object in integer form. The value received from the method is used as the bucket number. The bucket number is the address of the element inside the map. Hash code of null Key is 0.
- **Buckets:** Array of the node is called buckets. Each node has a data structure like a LinkedList. More than one node can share the same bucket. It may be different in capacity.

Insert Key, Value pair in HashMap

We use put() method to insert the Key and Value pair in the HashMap. The default size of HashMap is 16 (0 to 15).

Example

In the following example, we want to insert three (Key, Value) pair in the HashMap.

```
1. HashMap<String, Integer> map = new HashMap<>();
2. map.put("Aman", 19);
3. map.put("Sunny", 29);
4. map.put("Ritesh", 39);
```

Let's see at which index the Key, value pair will be saved into HashMap. When we call the put() method, then it calculates the hash code of the Key "Aman." Suppose the hash code of "Aman" is 2657860. To store the Key in memory, we have to calculate the index.

Calculating Index

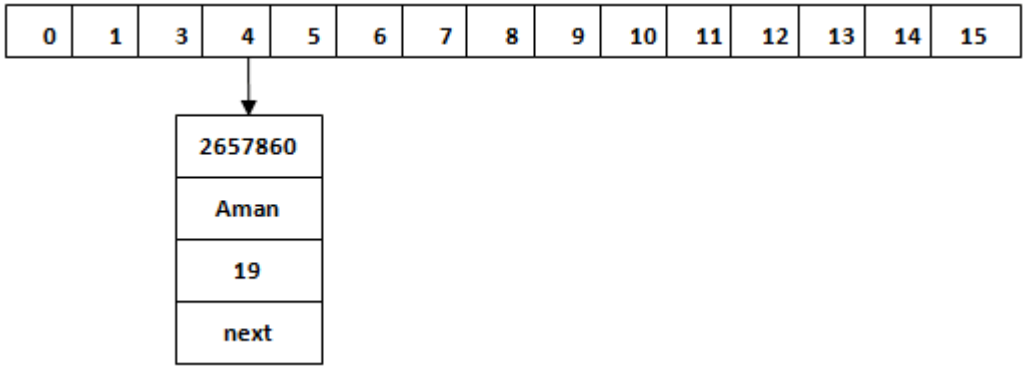
Index minimizes the size of the array. The Formula for calculating the index is:

```
1. Index = hashcode(Key) & (n-1)
```

Where n is the size of the array. Hence the index value for "Aman" is:

```
1. Index = 2657860 & (16-1) = 4
```

The value 4 is the computed index value where the Key and value will store in HashMap.

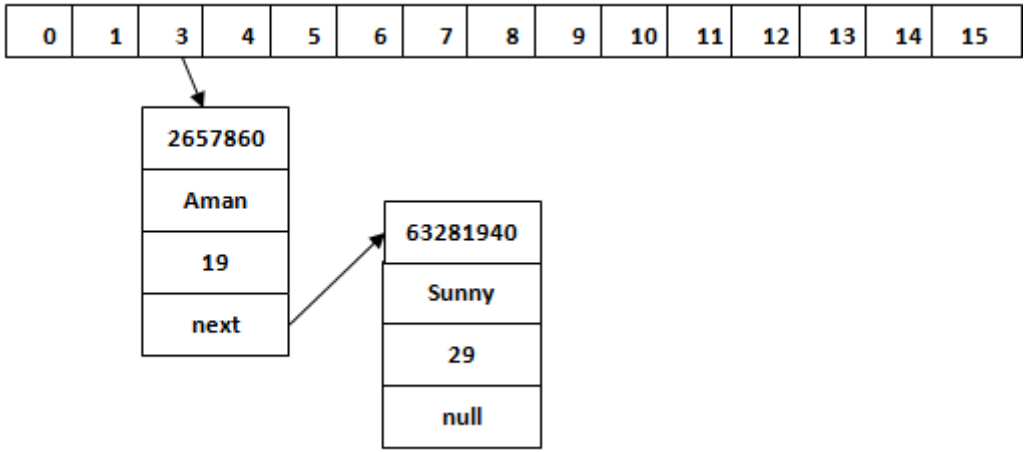


Hash Collision

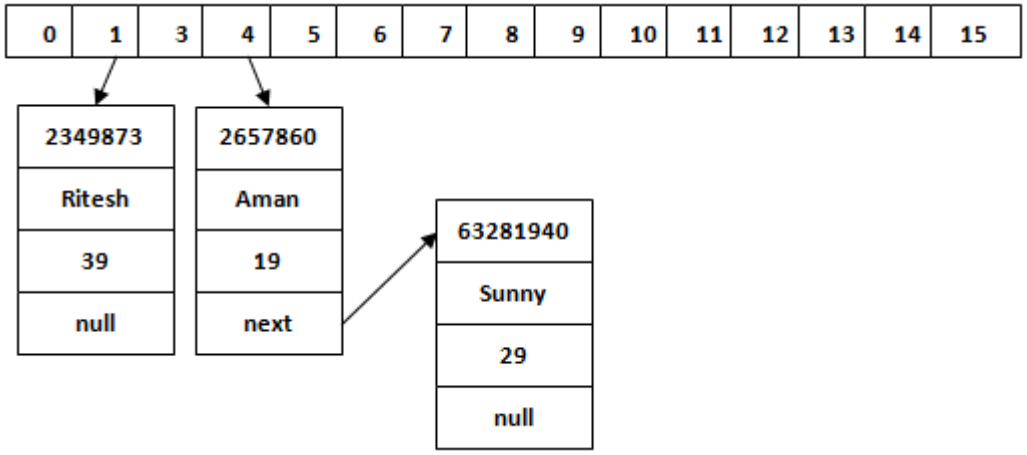
This is the case when the calculated index value is the same for two or more Keys. Let's calculate the hash code for another Key "Sunny." Suppose the hash code for "Sunny" is 63281940. To store the Key in the memory, we have to calculate index by using the index formula.

1. Index=63281940 & (16-1) = 4

The value 4 is the computed index value where the Key will be stored in HashMap. In this case, equals() method check that both Keys are equal or not. If Keys are same, replace the value with the current value. Otherwise, connect this node object to the existing node object through the LinkedList. Hence both Keys will be stored at index 4.



Similarly, we will store the Key "Ritesh." Suppose hash code for the Key is 2349873. The index value will be 1. Hence this Key will be stored at index 1.



get() method in HashMap

- get() method is used to get the value by its Key. It will not fetch the value if you don't know the Key. When get(K Key) method is called, it calculates the hash code of the Key.

- Suppose we have to fetch the Key "Aman." The following method will be called.

Difference between Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
4) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.

How to iterate Map in Java

In Java, iteration over Map can be done in various ways. Remember that we cannot iterate over map directly using **iterators**, because Map interface is not the part of Collection. All maps in Java implements **Map** interface. There are following types of maps in Java:

- HashMap
- TreeMap
- LinkedHashMap

A map is not a Collection but still, consider under the Collections framework. Hence, a Map is an interface which does not extend the Collections interface.

Iterator

An iterator is an interface used for iterate over a collection. It takes the place of Enumeration in Java Collections Framework. The difference between iterator and Enumeration is:

- The Iterator can traverse legacy and non-legacy elements whereas Enumeration can traverse only legacy elements.
- Iterator is fail-fast whereas Enumeration is not fail-fast.

Collection Views

The collection views method allows a map to be viewed as a Collection in the following ways:

- **keySet:** It is the set of keys contained in the Map.
- **values:** It is the collection of values contained in the Map.
- **entrySet:** It is a set of key-value pair in the Map.

The Map interface also has a small nested interface called **Map.entry**. The collection view provides the only means to iterate over a map.

Using Iterator interface

Example of iteration over HashMap

```
1.  public static void main(String args[])
2.  {
3.  HashMap<Integer, String> hm = new HashMap<Integer, String>();  //implements map interface
4.  hm.put(110,"Ravi");
5.  hm.put(120,"Prateek");
6.  hm.put(130, "Davesh");
7.  hm.put(140, "Kamal");
8.  hm.put(150, "Pawan");
9.  Iterator <Integer> it = hm.keySet().iterator();       //keyset is a method
10. while(it.hasNext())
11. {
12. int key=(int)it.next();
13. System.out.println("Roll no.: "+key+"     name: "+hm.get(key));
14. }
15. }
16. }
```

Output:

```
Roll no.: 130     name: Davesh
Roll no.: 150     name: Pawan
Roll no.: 120     name: Prateek
Roll no.: 140     name: Kamal
Roll no.: 110     name: Ravi
```

The get() method of Map interface in Java is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.

Syntax:

Parameter: The method takes one parameter *key_element* of object type and refers to the key whose associated value is supposed to be fetched.
Return Value: The method returns the value associated with the *key_element* in this Map collection.
Below programs illustrates the working of java.util.Map.get() method:

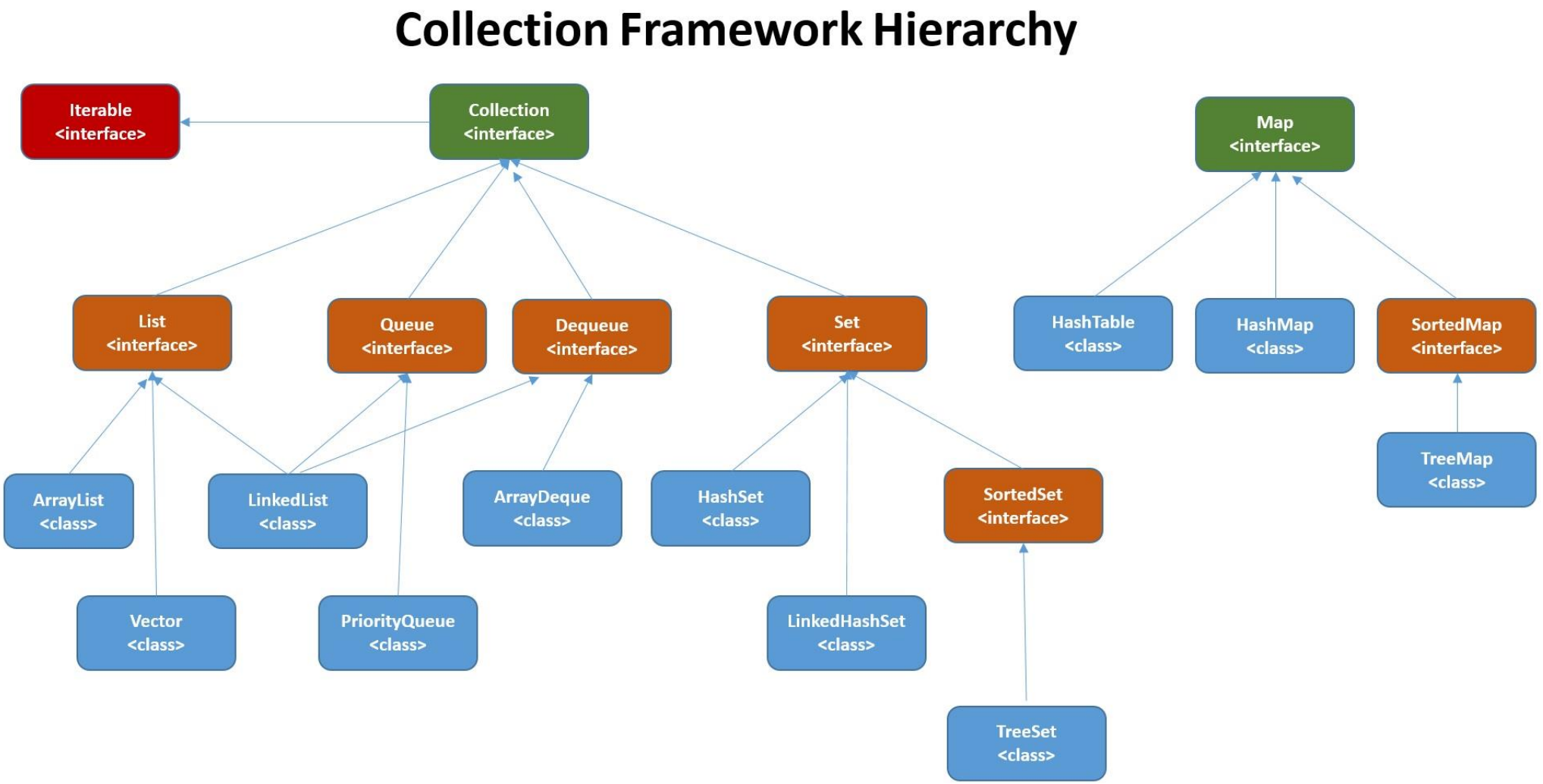
Program 1: Mapping String Values to Integer Keys.

```
// Java code to illustrate the get() method
import java.util.*;

public class Map_Demo {
    public static void main(String[] args)
    {
        // Creating an empty Map
        Map<Integer, String> map = new HashMap<Integer, String>();

        // Mapping string values to int keys
        map.put(10, "Geeks");
        map.put(15, "4");
        map.put(20, "Geeks");
        map.put(25, "Welcomes");
        map.put(30, "You");
```

HashMap vs HashSet:

Parameter	HashMap	HashSet
Interface	This is core difference among them.HashMap implements Map interface	HashSet implement Set interface
Method for storing data	It stores data in a form of key->value pair.So it uses put(key,value) method for storing data	It uses add(value) method for storing data
Duplicates	HashMap allows duplicate value but not duplicate keys	HashSet does not allow duplicate values.
Performance	It is faster than hashset as values are stored with unique keys	It is slower than HashMap
HashCode Calculation	In hash map hashcode value is calculated using key object	In this,hashcode is calculated on the basis of value object.Hashcode can be same for two value object so we have to implement equals() method.If equals() method return false then two objects are different.

hierarchy of collection



diff bet map and collection?

Collection and Map both are interfaces in java. util package but **Collection is used to store objects and Map is used to store the objects for (key,value) based manner**. Collection classes are used to store object in array format. and Map classes are used to store in (KEY,VALUE) pair format.

ArrayList	HashSet
ArrayList implements List interface	HashSet implements Set interface
ArrayList allow duplicates	HashSet doesn't allow duplicates
ArrayList is an ordered collection and maintains insertion order of elements	HashSet is an unordered collection and doesn't maintain any order
ArrayList is backed by an Array	HashSet is backed by an HashMap instance.
In ArrayList using index we can retrieve object by calling get(index) or remove objects by calling remove(index)	HashSet is completely object based. HashSet also doesn't provide get() method

What is HashMap and its applications?

HashMap <K V> in java. util package implements the Map interface in java collection based on Hashtable, where K is the type of the keys and V is the type of the mapped value. It **allows users to store key-value pair, where no duplicate keys are allowed**. Order is not maintained as well.

HashMap	Hashtable
HashMap is **non synchronized**. It is **not-thread safe** and can't be shared between many threads without proper synchronization code.	Hashtable is **synchronized**. It is **thread-safe** and can be shared with many threads
HashMap **allows one null key and multiple null values**.	Hashtable **doesn't allow any null key or value**.
HashMap is a **new class introduced in JDK 1.2**.	Hashtable is a **legacy class**
HashMap is much faster and uses less memory than Hashtable **Unsynchronized objects are often much better in performance in compare to synchronized object**	Hashtable is **slow**.
HashMap is **traversed by Iterator**.	Hashtable is **traversed by Enumerator and Iterator**.
Iterator in HashMap is **fail-fast**.	Enumerator in Hashtable is **not fail-fast**.
HashMap inherits **AbstractMap** class.	Hashtable inherits **Dictionary** class.

How get mehod will callin hashmap?

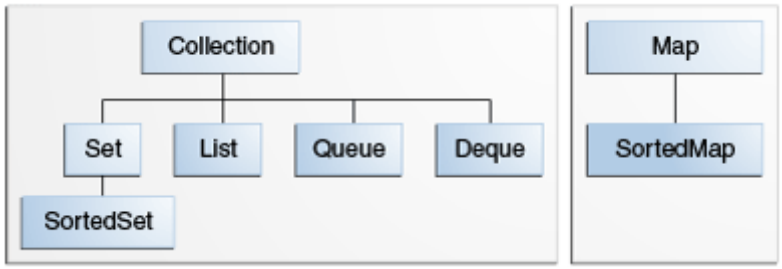
HashMap get() Method in Java
get() method of HashMap class is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.

What is the syntax for iterating hashamap?

keyset(): A keySet() method of HashMap class is used for iteration over the keys contained in the map. It returns the Set view of the keys. values(): A values() method of HashMap class is used for iteration over the values contained in the map. It returns a collection view of the values.

What are interfaces in collection?

The Collection interface is **the least common denominator that all collections implement and is used to pass collections around and to manipulate them when maximum generality is desired**. Some types of collections allow duplicate elements, and others do not. Some are ordered and others are unordered.



List	Set
List interface allows duplicate elements	Set does not allows duplicate elements
List maintains insertion order	But Set do not maintains any insertion order
List implementation classes are ArrayList,LinkedList .	Set implementation classes are HashSet,LinkedHashSet and TreeSet.
Vector is only the legacy class	No legacy classes
We can add any number of null values	But in Set almost only one null value
To traverse the List elements by using ListIrerator	Iterator can be used traverse the Set elements .

Algoritham used in hash collision ?

A collision, or more specifically, a hash code collision in a HashMap, is **a situation where two or more key objects produce the same final hash value and hence point to the same bucket location or array index**.

What is the concurrent hashmap exception?

The ConcurrentModificationException **occurs when an object is tried to be modified concurrently when it is not permissible**. This exception usually comes when one is working with Java Collection classes. For Example - It is not permissible for a thread to modify a Collection when some other thread is iterating over it.

ConcurrentModificationException in Java

The ConcurrentModificationException occurs when an object is tried to be modified concurrently when it is not permissible. This exception usually comes when one is working with **Java Collection classes**.

For Example - It is not permissible for a thread to modify a Collection when some other thread is iterating over it. This is because the result of the iteration becomes undefined with it. Some implementation of the Iterator class throws this exception, including all those general-purpose implementations of Iterator which are provided by the JRE. Iterators which do this are called **fail-fast** as they throw the exception quickly as soon as they encounter such situation rather than facing undetermined behavior of the collection any time in the future.

Working of HashSet in Java

Java Set Interface

A Java Set interface represents a group of elements arranged like an array. It does not allow duplicate elements. When we try to pass the same element that is already available in the Set, then it will not store into the Set. It is used to model the mathematical set abstraction.

Java HashSet class

A Java HashSet class represents a set of elements (objects). It does not guarantee the order of elements. It constructs a collection that uses a hash table for storing elements. It contains unique elements. It inherits the AbstractSet class. It also implements the Set interface. It uses a technique to store elements is called hashing. HashSet uses HashMap internally in Java.

Suppose, we want to create a HashSet to store a group of Strings, then create the object as:

```
1. HashSet<String> hs=new HashSet<>();
```

Where <String> is the generic type parameter. It represents the type of element storing in the HashSet.

HashSet implements Set interface. It guarantees uniqueness. It is achieved by storing elements as keys with the same value always. HashSet does not have any method to retrieve the object from the HashSet. There is only a way to get objects from the HashSet via Iterator. When we create an object of HashSet, it internally creates an instance of HashMap with default initial capacity 16.

HashSet uses a constructor HashSet(int capacity) that represents how many elements can be stored in the HashSet. The capacity may increase automatically when more elements to be store.

HashSet uses another constructor HashSet(int capacity, float loadfactor). Here, loadfactor demines the point where the capacity of HashSet would be increased internally. For example, the product of capacity and loadfactor is 101*0.5=50.5. It means that after storing 50th element into the HashSet; its capacity will be internally increased to store more elements. The initial default capacity of HashSet is 16. The default load factor is 0.75.

While iterating hashamap if we add another object what will be the output?

HashMap is a part of Java's collection providing the basic implementation of the Map interface of Java by storing the data in (Key, Value) pairs to access them by an index of another type. One object is listed as a key (index) to another object (value). If you try to insert the duplicate key, it will replace the element of the corresponding key. In order to use this class and its methods, it is necessary to import *java.util.HashMap* package or its superclass.
There is a numerous number of ways to iterate over HashMap of which 5 are listed as below:

1. Iterate through a HashMap EntrySet using Iterators.
2. Iterate through HashMap KeySet using Iterator.
3. Iterate HashMap using for-each loop.
4. Iterating through a HashMap using Lambda Expressions.
5. Loop through a HashMap using Stream API.

Method 1: Using a for loop to iterate through a HashMap. Iterating a HashMap through a for loop to use *getValue()* and *getKey()* functions.
Implementation: In the code given below, entrySet() is used to return a set view of mapped elements. From the code given below:

- set.getValue() to get value from the set.
- set.getKey() to get key from the set.

Hashmap is thread safe or not?

And, importantly, **HashMap is not a thread-safe implementation**, while Hashtable does provide thread-safety by synchronizing operations. Even though Hashtable is thread safe, it is not very efficient. Another fully synchronized Map, Collections.

Locking mechanism in concurrent hashmap collection?

In ConcurrentHashMap, at a time any number of threads can perform retrieval operation but for updated in the object, the thread must lock the particular segment in which the thread wants to operate. This type of locking mechanism is known as **Segment locking or bucket locking**.

How we can remove arraylist dupliates if w don't know any value or index of that element?

Using Iterator

Approach:

1. Get the ArrayList with duplicate values.
2. Create another ArrayList.
3. Traverse through the first arraylist and store the first appearance of each element into the second arraylist using contains() method.
4. The second ArrayList contains the elements with duplicates removed.

Using LinkedHashSet

A better way (both time complexity and ease of implementation wise) is to remove duplicates from an ArrayList is to convert it into a Set that does not allow duplicates. Hence LinkedHashSet is the best option available as this do not allows duplicates as well it preserves the insertion order.

Approach:

1. Get the ArrayList with duplicate values.
2. Create a LinkedHashSet from this ArrayList. This will remove the duplicates
3. Convert this LinkedHashSet back to Arraylist.
4. The second ArrayList contains the elements with duplicates removed.

Using Java 8 Stream.distinct()

You can use the distinct() method from the Stream API. The distinct() method return a new Stream without duplicates elements based on the result returned by equals() method, which can be used for further processing. The actual processing of Stream pipeline starts only after calling terminal methods like forEach() or collect().

Approach:

1. Get the ArrayList with duplicate values.
2. Create a new List from this ArrayList.
3. Using Stream().distinct() method which return distinct object stream.
4. convert this object stream into List

5. Null insertion is possible for hashmap ?

HashMap: HashMap implements all of the Map operations and **allows null values and one null key**. HashMap does not maintain an order of its key-value elements. Therefore, consider to use a HashMap when order does not matter and nulls are acceptable.

Why does ConcurrentHashMap does not allow null key or values?

The main reason that null is not allowed in ConcurrentMaps such as ConcurrentHashMaps, ConcurrentSkipListMaps is to avoid ambiguities. If map.get(key) returns null, you cannot detect whether the key explicitly maps to null or the key itself is not mapped. In a non-concurrent map, you may check this using map.contains(key), but in a concurrent environment, the map might have changed between calls.

Fail fast and fail safe iterator?

Fail Fast and Fail Safe Iterator in Java

Iterators in Java are part of the Java Collection framework. They are used to retrieve elements one by one. The Java Collection supports two types of iterators; Fail Fast and Fail Safe. These iterators are very useful in exception handling.

The Fail fast iterator aborts the operation as soon it exposes failures and stops the entire operation. Comparatively, Fail Safe iterator doesn't abort the operation in case of a failure. Instead, it tries to avoid failures as much as possible.

In this section, we will discuss both the fail fast and Fail Safe iterators with examples. Further, we will see the differences between them.

Before understanding the Fail fast and Fail Safe iterator, let's understand the concurrent modification:

Concurrent Modification

The Concurrent modification in Java is to modify an object concurrently while another task is running over it. In simple terms, concurrent modification is the process of modifying objects while another thread is running over them. It will change the structure of the data collection, either by removing, adding, or updating the value of the elements in the collection.

Not all iterators support this behavior; implementation of some iterator may throw ConcurrentModificationException.

Let's understand the Fail Fast and Fail Safe systems:

Fail Fast and Fail Safe Systems

The Fail Fast system is a system that shuts down immediately after an error is reported. All the operations will be aborted instantly in it.

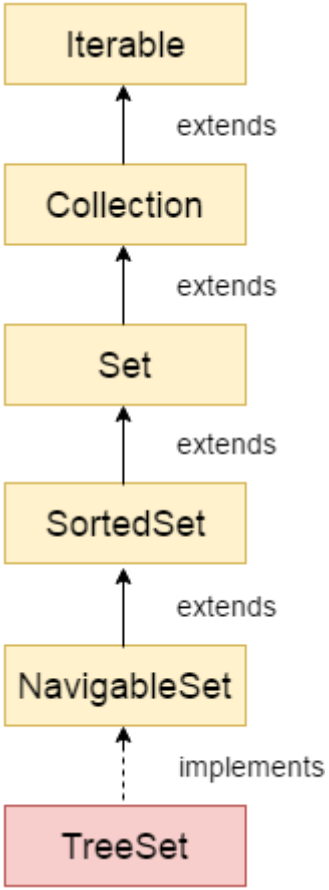
The Fail Safe is a system that continues to operate even after an error or fail has occurred. These systems do not abort the operations instantly; instead, they will try to hide the errors and will try to avoid failures as much as possible.

Fail Fast Iterator

The iterator in Java is used to traverse over a collection's objects. The collections return two types of iterators, either it will be Fail Fast or Fail Safe.

The Fail Fast iterators immediately throw ConcurrentModificationException in case of structural modification of the collection. Structural modification means adding, removing, updating the value of an element in a data collection while another thread is iterating over that collection. Some examples of Fail Fast iterator are iterator on ArrayList, HashMap collection classes.

Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about the Java TreeSet class are:

- Java TreeSet class contains unique elements only like HashSet.

- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quite fast.
- Java TreeSet class doesn't allow null elements.
- Java TreeSet class is non-synchronized.
- Java TreeSet class maintains ascending order.
- The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

Internal Working of The TreeSet Class

TreeSet is being implemented using a binary search tree, which is self-balancing just like a Red-Black Tree. Therefore, operations such as a search, remove, and add consume O(log(N)) time. The reason behind this is there in the self-balancing tree. It is there to ensure that the tree height never exceeds O(log(N)) for all of the mentioned operations. Therefore, it is one of the efficient data structures in order to keep the large data that is sorted and also to do operations on it.

Synchronization of The TreeSet Class

As already mentioned above, the TreeSet class is not synchronized. It means if more than one thread concurrently accesses a tree set, and one of the accessing threads modify it, then the synchronization must be done manually. It is usually done by doing some object synchronization that encapsulates the set. However, in the case where no such object is found, then the set must be wrapped with the help of the Collections.synchronizedSet() method. It is advised to use the method during creation time in order to avoid the unsynchronized access of the set. The following code snippet shows the same.

OOPs concept

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Polymorphisam?

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object

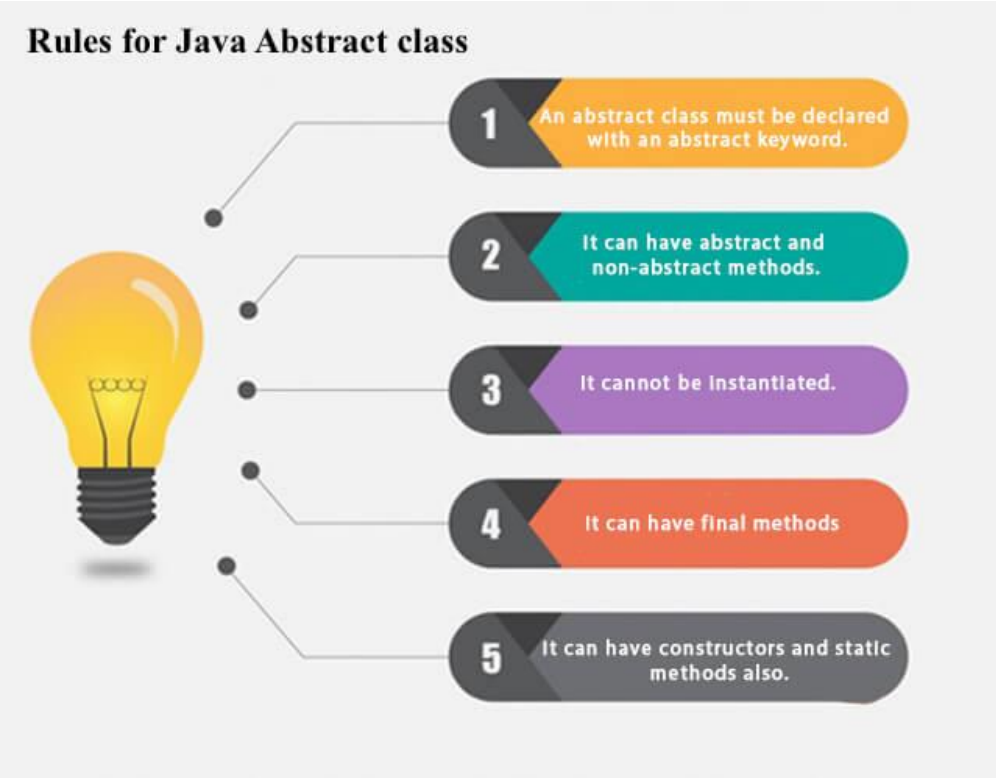
Abstract Classes and Methods

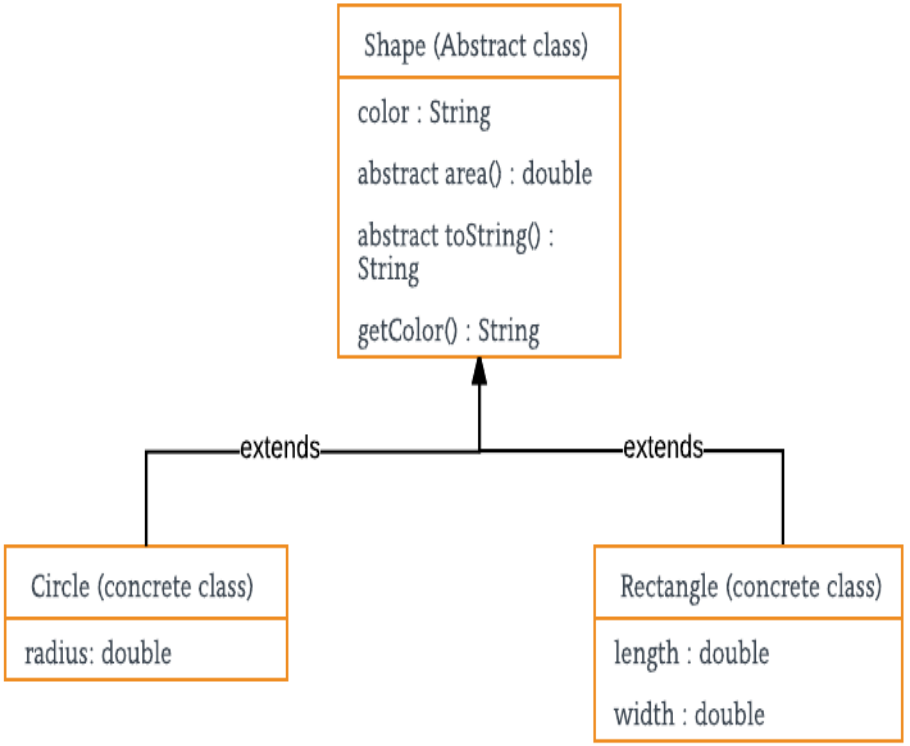
Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The `abstract` keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).


Rules for Java Abstract class
1
An abstract class must be declared with an abstract keyword.
2
It can have abstract and non-abstract methods.
3
It cannot be instantiated.
4
It can have final methods
5
It can have constructors and static methods also.


Shape (Abstract class)
color : String
abstract area() : double
abstract toString() : String
getColor() : String
extends
extends
Circle (concrete class)
radius: double
Rectangle (concrete class)
length : double
width : double

OOPs interface vs abstract class

Interface	Abstract class
Interface support multiple implementations.	Abstract class does not support multiple inheritance.
Interface does not contain Data Member	Abstract class contains Data Member
Interface does not contain Constructors	Abstract class contains Constructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

No.	Method Overloading	Method Overriding
1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

The Four Pillars of Object-Oriented Programming

- Abstraction.
- Encapsulation.
- Inheritance.
- Polymorphism.

Aggregation	Composition
Aggregation is a weak Association.	Composition is a strong Association.
Class can exist independently without owner.	Class can not meaningfully exist without owner.
Have their own Life Time.	Life Time depends on the Owner.
A uses B.	A owns B.
Child is not owned by 1 owner.	Child can have only 1 owner.
Has-A relationship. A has B.	Part-Of relationship. B is part of A.
Denoted by a empty diamond in UML.	Denoted by a filled diamond in UML.
We do not use "final" keyword for Aggregation.	"final" keyword is used to represent Composition.
Examples: - Car has a Driver. - A Human uses Clothes. - A Company is an aggregation of People. - A Text Editor uses a File. - Mobile has a SIM Card.	Examples: - Engine is a part of Car. - A Human owns the Heart. - A Company is a composition of Accounts. - A Text Editor owns a Buffer. - IMEI Number is a part of a Mobile.

can we access properties of parent and why?

You can do it by typecasting the reference to the child type. Show activity on this post. The reason it isn't working is because parent classes don't have access to their childrens properties. You may be creating a Child class, but you assign it to a Parent object.

Inheritance?

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

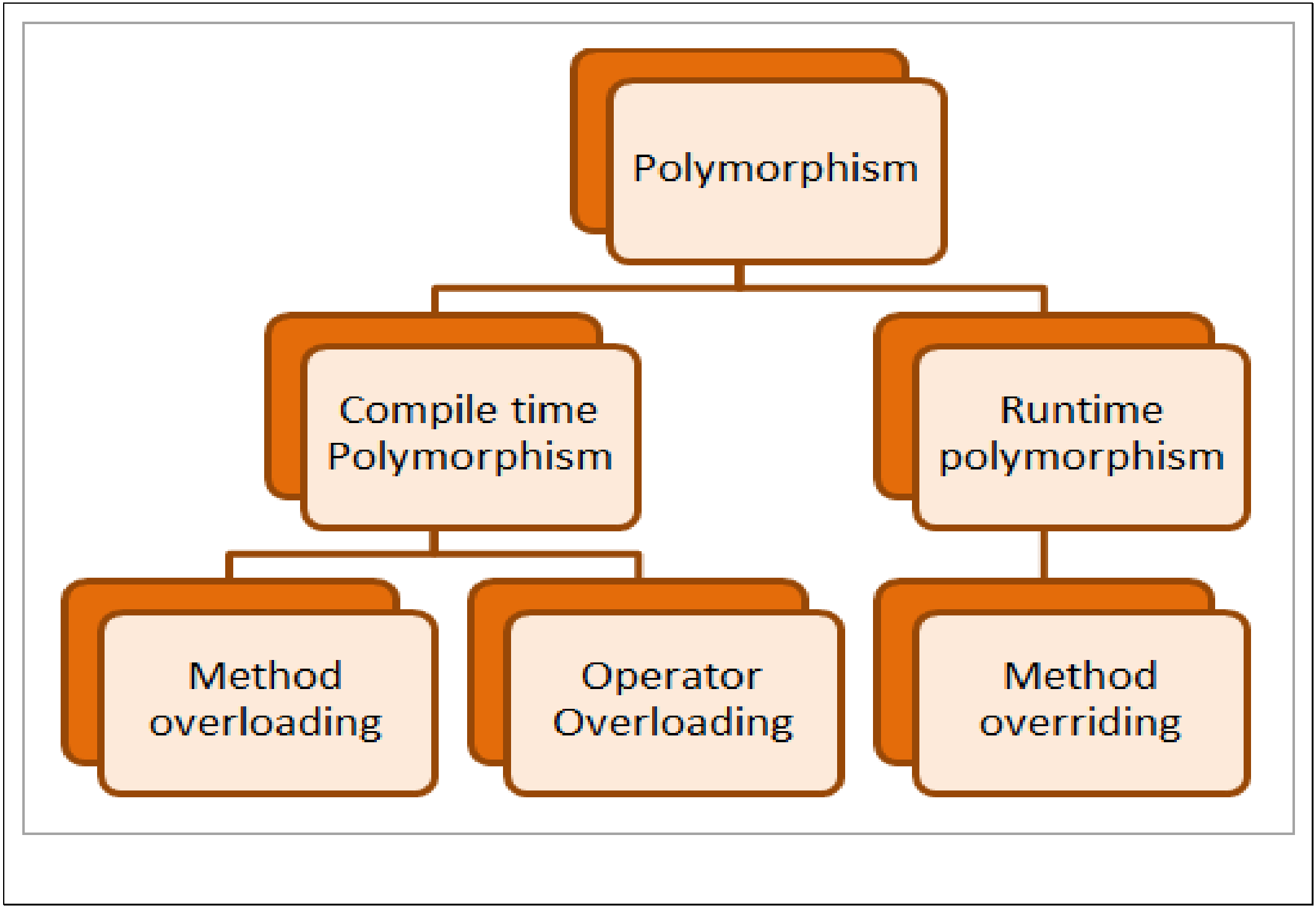
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

dynamic polymorphism?

Dynamic polymorphism is **a process or mechanism in which a call to an overridden method is to resolve at runtime rather than compile-time**. It is also known as runtime polymorphism or dynamic method dispatch. We can achieve dynamic polymorphism by using the method overriding.

what is loose coupling?

Loose coupling in Java means that **the classes are independent of each other**. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling. If a situation requires objects to be used from outside, it is termed as a loose coupling situation.


Polymorphism
Compile time Polymorphism
Runtime polymorphism
Method overloading
Operator Overloading
Method overriding

OBJECT	CLASS
Object is an instance ofa class.	Class is a blue print from which objects are created
Object is a real world entity such as chair, pen. table, laptop etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
Object is created through new keyword. Employee ob = new Employee();	Class is declared using class keyword. class Employee{}
There are different ways to create object in java:- New keyword, newinstance() method, clone() method, And deserialization.	There is only one way to define a class, i.e., by using class keyword.

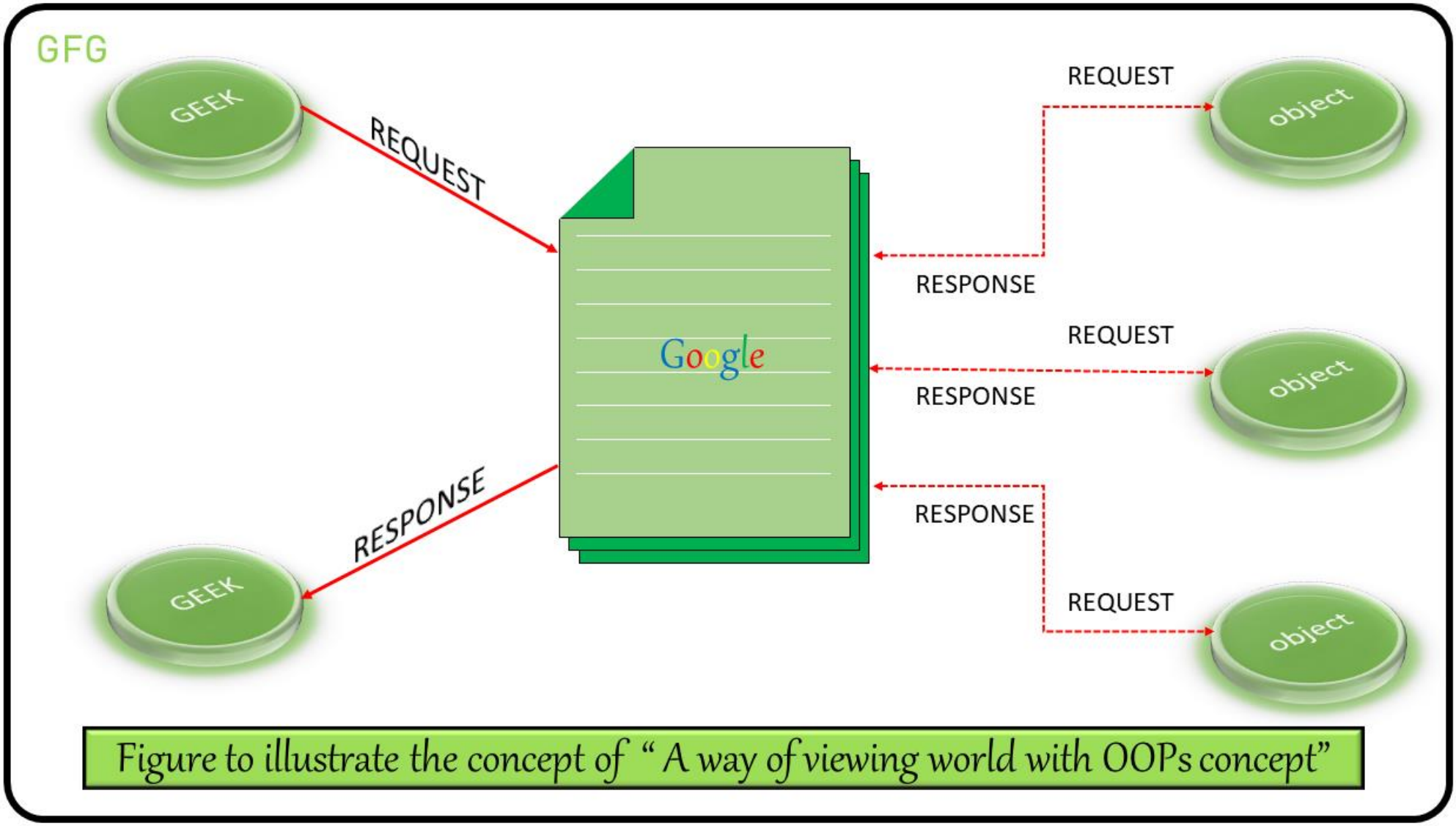


Figure to illustrate the concept of " A way of viewing world with OOPs concept"

Method Overriding in Java?

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

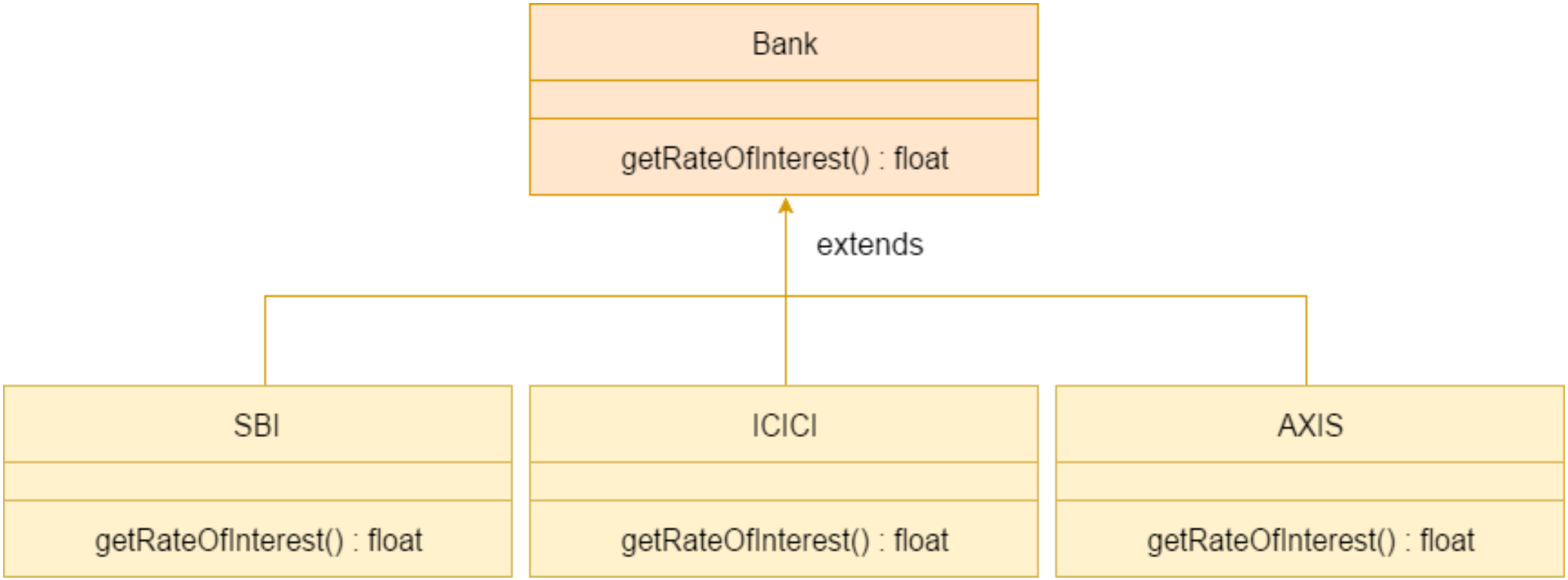
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Method Overloading in Java?

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Why we use super keyword in constructor?

The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is **to eliminate the confusion between superclasses and subclasses that have methods with the same name**.

Diamond problem In java?

The Diamond Problem

The diamond problem is a common problem in Java when it comes to inheritance. Inheritance is a very popular property in an object-oriented programming language, such as C++, Java, etc. There are different types of inheritance such as, single, multiple, multi-level, and hybrid inheritance. But remember that **Java does not support the multiple inheritance** because of the diamond problem.

```
/D.java:25: error: '{' expected
public class D extends B,C
                         ^
1 error
```

As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super-class A, then it is called simple inheritance, and Java supports them.

Multi-level inheritance allows a child class to inherit properties from a class that can inherit properties from some other classes. For example, class C can inherit its property from B class which itself inherits from A class. Java also supports them.

What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**. In the above figure, we find that class D is trying to inherit form class B and class C, that is not allowed in Java.

It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPPs languages. It is sometimes referred to as the **deadly diamond of death**.

The Solution of Diamond Problem

The solution to the diamond problem is **default methods** and **interfaces**. We can achieve multiple inheritance by using these two things.

The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.

The advantage of interfaces is that it can have the same default methods with the same name and signature in two different interfaces. It allows us to implement these two interfaces, from a class. We must override the default methods explicitly with its interface name.

When we make constructor as private the what will be the effect of private constructor?

Introduction. Private constructors allow us to restrict the instantiation of a class. Simply put, they **prevent the creation of class instances in any place other than the class itself**.

Which constructor is called automatically?

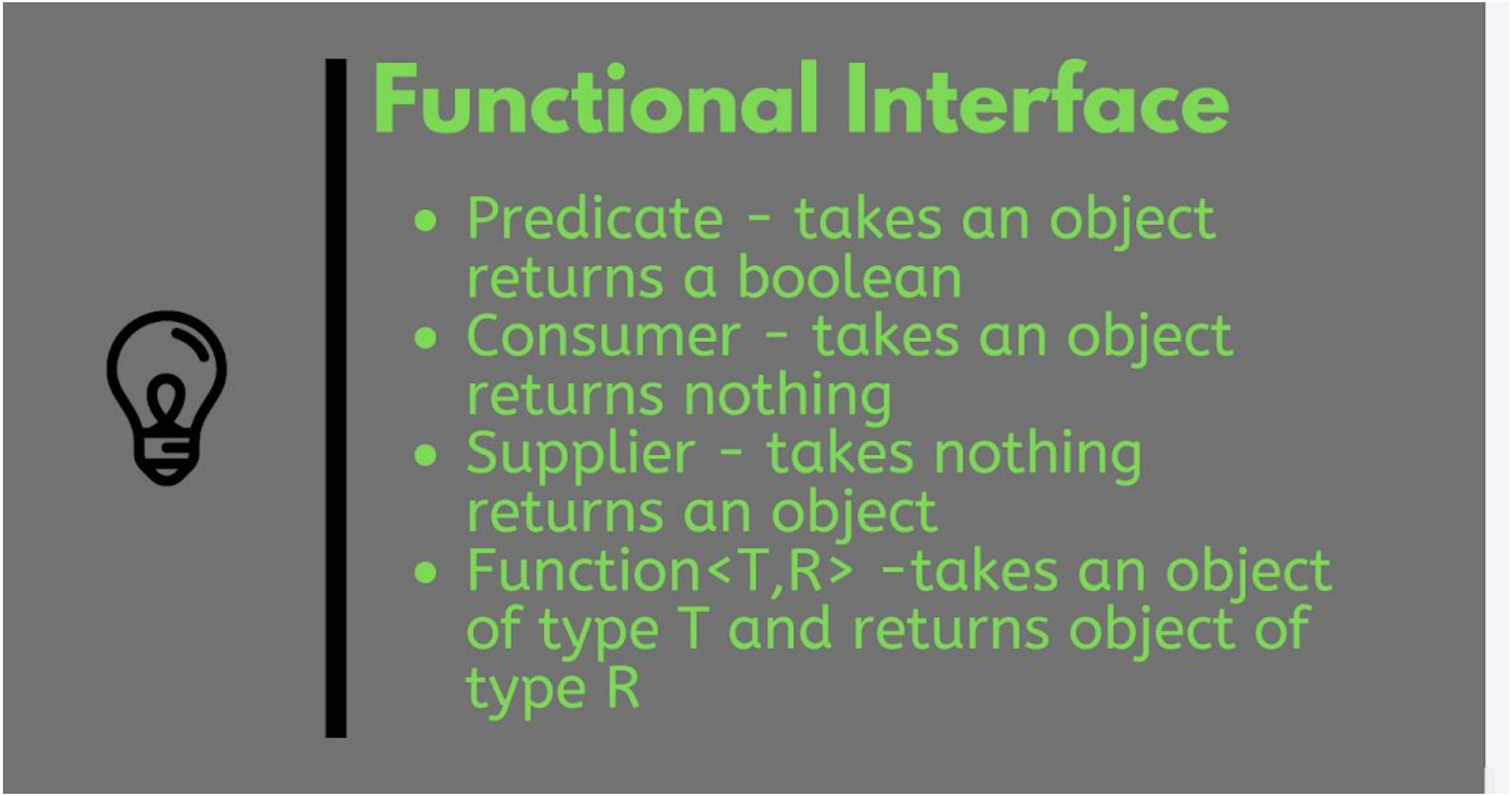
If members have **default constructors or constructor without parameter** then these constructors are called automatically, otherwise parameterized constructors can be called using Initializer List

What happens if constructor is protected in Java?

Protecting a constructor **prevents the users from creating the instance of the class, outside the package**. During overriding, when a variable or method is protected, it can be overridden to other subclass using either a public or protected modifier only. Outer class and interface cannot be protected.

What happen when we make class as private?

Private classes are allowed, but only as inner or nested classes. If you have a private inner or nested class, then **access is restricted to the scope of that outer class**. If you have a private class on its own as a top-level class, then you can't get access to it from anywhere.



Java Functional Interfaces

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.

Functional Interface is also known as Single Abstract Method Interfaces or SAM Interfaces. It is a new feature in Java, which helps to achieve functional programming approach.

```
1.  @FunctionalInterface
2.  interface sayable{
3.      void say(String msg);
4.  }
5.  public class FunctionalInterfaceExample implements sayable{
6.      public void say(String msg){
7.          System.out.println(msg);
8.      }
9.      public static void main(String[] args) {
10.         FunctionalInterfaceExample fie = new FunctionalInterfaceExample();
11.         fie.say("Hello there");
12.     }
```

Java Predefined-Functional Interfaces

Java provides predefined functional interfaces to deal with functional programming by using lambda and method references.

You can also define your own custom functional interface. Following is the list of functional interface which are placed in java.util.function package.

Interface	Description

BiConsumer<T,U>	It represents an operation that accepts two input arguments and returns no result.
Consumer<T>	It represents an operation that accepts a single argument and returns no result.
Function<T,R>	It represents a function that accepts one argument and returns a result.
Predicate<T>	It represents a predicate (boolean-valued function) of one argument.
BiFunction<T,U,R>	It represents a function that accepts two arguments and returns a a result.
BinaryOperator<T>	It represents an operation upon two operands of the same data type. It returns a result of the same type as the operands.
BiPredicate<T,U>	It represents a predicate (boolean-valued function) of two arguments.
BooleanSupplier	It represents a supplier of boolean-valued results.
DoubleBinaryOperator	It represents an operation upon two double type operands and returns a double type value.
DoubleConsumer	It represents an operation that accepts a single double type argument and returns no result.
DoubleFunction<R>	It represents a function that accepts a double type argument and produces a result.
DoublePredicate	It represents a predicate (boolean-valued function) of one double type argument.
DoubleSupplier	It represents a supplier of double type results.

DoubleToIntFunction	It represents a function that accepts a double type argument and produces an int type result.
DoubleToLongFunction	It represents a function that accepts a double type argument and produces a long type result.
DoubleUnaryOperator	It represents an operation on a single double type operand that produces a double type result.
IntBinaryOperator	It represents an operation upon two int type operands and returns an int type result.
IntConsumer	It represents an operation that accepts a single integer argument and returns no result.
IntFunction<R>	It represents a function that accepts an integer argument and returns a result.
IntPredicate	It represents a predicate (boolean-valued function) of one integer argument.
IntSupplier	It represents a supplier of integer type.
IntToDoubleFunction	It represents a function that accepts an integer argument and returns a double.
IntToLongFunction	It represents a function that accepts an integer argument and returns a long.
IntUnaryOperator	It represents an operation on a single integer operand that produces an integer result.
LongBinaryOperator	It represents an operation upon two long type operands and returns a long type result.
LongConsumer	It represents an operation that accepts a single long type argument and returns no result.
LongFunction<R>	It represents a function that accepts a long type argument and returns a result.

LongPredicate	It represents a predicate (boolean-valued function) of one long type argument.
LongSupplier	It represents a supplier of long type results.
LongToDoubleFunction	It represents a function that accepts a long type argument and returns a result of double type.
LongToIntFunction	It represents a function that accepts a long type argument and returns an integer result.
LongUnaryOperator	It represents an operation on a single long type operand that returns a long type result.
ObjDoubleConsumer<T>	It represents an operation that accepts an object and a double argument, and returns no result.
ObjIntConsumer<T>	It represents an operation that accepts an object and an integer argument. It does not return result.
ObjLongConsumer<T>	It represents an operation that accepts an object and a long argument, it returns no result.
Supplier<T>	It represents a supplier of results.
ToDoubleBiFunction<T,U>	It represents a function that accepts two arguments and produces a double type result.
ToDoubleFunction<T>	It represents a function that returns a double type result.
ToIntBiFunction<T,U>	It represents a function that accepts two arguments and returns an integer.
ToIntFunction<T>	It represents a function that returns an integer.
ToLongBiFunction<T,U>	It represents a function that accepts two arguments and returns a result of long type.

ToLongFunction<T>	It represents a function that returns a result of long type.
UnaryOperator<T>	It represents an operation on a single operand that returnsa a result of the same type as its operand.

Java 8 Stream API

Java provides a new additional package in Java 8 called java.util.stream. This package consists of classes, interfaces and enum to allows functional-style operations on the elements. You can use stream by importing java.util.stream package.

Stream 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.

You can use stream to filter, collect, print, and convert from one data structure to other etc. In the following examples, we have apply various operations with the help of stream.

Java Stream Interface Methods

Methods	Description
boolean allMatch(Predicate<? super T> predicate)	It returns all elements of this stream which match the provided predicate. If the stream is empty then true is returned and the predicate is not evaluated.
boolean anyMatch(Predicate<? super T> predicate)	It returns any element of this stream that matches the provided predicate. If the stream is empty then false is returned and the predicate is not evaluated.
static <T> Stream.Builder<T> builder()	It returns a builder for a Stream.
<R,A> R collect(Collector<? super T,A,R> collector)	It performs a mutable reduction operation on the elements of this stream using a Collector. A Collector encapsulates the functions used as arguments to collect(Supplier, BiConsumer, BiConsumer), allowing for reuse of collection strategies and composition of collect operations such as multiple-level grouping or partitioning.
<R> R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)	It performs a mutable reduction operation on the elements of this stream. A mutable reduction is one in which the reduced value is a mutable result container, such as an ArrayList, and elements are incorporated by updating the state of the result rather than by replacing the result.
static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)	It creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream. The resulting stream is ordered if both of the input streams are ordered, and parallel if either of the input streams is parallel. When the resulting stream is closed, the close handlers for both input streams are invoked.

long count()	It returns the count of elements in this stream. This is a special case of a reduction.
Stream<T> distinct()	It returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
static <T> Stream<T> empty()	It returns an empty sequential Stream.
Stream<T> filter(Predicate<? super T> predicate)	It returns a stream consisting of the elements of this stream that match the given predicate.
Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)	It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)	It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)

IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)	It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
void forEach(Consumer<? super T> action)	It performs an action for each element of this stream.
void forEachOrdered(Consumer<? super T> action)	It performs an action for each element of this stream, in the encounter order of the stream if the stream has a defined encounter order.
static <T> Stream<T> generate(Supplier<T> s)	It returns an infinite sequential unordered stream where each element is generated by the provided Supplier. This is suitable for generating constant streams, streams of random elements, etc.
static <T> Stream<T> iterate(T seed,UnaryOperator<T> f)	It returns an infinite sequential ordered Stream produced by iterative application of a function f to an initial element seed, producing a Stream consisting of seed, f(seed), f(f(seed)), etc.
Stream<T> limit(long maxSize)	It returns a stream consisting of the elements of this stream, truncated to be no longer than maxSize in length.
<R> Stream<R> map(Function<? super T,?	It returns a stream consisting of the results of applying the given function to

extends R> mapper)	the elements of this stream.
DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)	It returns a DoubleStream consisting of the results of applying the given function to the elements of this stream.
IntStream mapToInt(ToIntFunction<? super T> mapper)	It returns an IntStream consisting of the results of applying the given function to the elements of this stream.
LongStream mapToLong(ToLongFunction<? super T> mapper)	It returns a LongStream consisting of the results of applying the given function to the elements of this stream.
Optional<T> max(Comparator<? super T> comparator)	It returns the maximum element of this stream according to the provided Comparator. This is a special case of a reduction.
Optional<T> min(Comparator<? super T> comparator)	It returns the minimum element of this stream according to the provided Comparator. This is a special case of a reduction.
boolean noneMatch(Predicate<? super T> predicate)	It returns elements of this stream match the provided predicate. If the stream is empty then true is returned and the predicate is not evaluated.
@SafeVarargs static <T> Stream<T> of(T... values)	It returns a sequential ordered stream whose elements are the specified values.
static <T> Stream<T> of(T t)	It returns a sequential Stream containing a single element.
Stream<T> peek(Consumer<? super T> action)	It returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed

	from the resulting stream.
Optional<T> reduce(BinaryOperator<T> accumulator)	It performs a reduction on the elements of this stream, using an associative accumulation function, and returns an Optional describing the reduced value, if any.
T reduce(T identity, BinaryOperator<T> accumulator)	It performs a reduction on the elements of this stream, using the provided identity value and an associative accumulation function, and returns the reduced value.
<U> U reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)	It performs a reduction on the elements of this stream, using the provided identity, accumulation and combining functions.
Stream<T> skip(long n)	It returns a stream consisting of the remaining elements of this stream after discarding the first n elements of the stream. If this stream contains fewer than n elements then an empty stream will be returned.
Stream<T> sorted()	It returns a stream consisting of the elements of this stream, sorted according to natural order. If the elements of this stream are not Comparable, a java.lang.ClassCastException may be thrown when the terminal operation is executed.
Stream<T> sorted(Comparator<? super T> comparator)	It returns a stream consisting of the elements of this stream, sorted according to the provided Comparator.
Object[] toArray()	It returns an array containing the elements of this stream.
<A> A[] toArray(IntFunction<A[]> generator)	It returns an array containing the elements of this stream, using the provided

	generator function to allocate the returned array, as well as any additional arrays that might be required for a partitioned execution or for resizing.

Java Example: Filtering Collection without using Stream

In the following example, we are filtering data without using stream. This approach we are used before the stream package was released.

```
1.  import java.util.*;
2.  class Product{
3.      int id;
4.      String name;
5.      float price;
6.      public Product(int id, String name, float price) {
7.          this.id = id;
8.          this.name = name;
9.          this.price = price;
10.     }
11. }
12. public class JavaStreamExample {
13.     public static void main(String[] args) {
14.         List<Product> productsList = new ArrayList<Product>();
15.         //Adding Products
16.         productsList.add(new Product(1,"HP Laptop",25000f));
17.         productsList.add(new Product(2,"Dell Laptop",30000f));
18.         productsList.add(new Product(3,"Lenevo Laptop",28000f));
19.         productsList.add(new Product(4,"Sony Laptop",28000f));
20.         productsList.add(new Product(5,"Apple Laptop",90000f));
```

```
21.         List<Float> productPriceList = new ArrayList<Float>();
22.        for(Product product: productsList){
23.
24.            // filtering data of list
25.             if(product.price<30000){
26.                 productPriceList.add(product.price);    // adding price to a productPriceList
27.             }
28.        }
29.         System.out.println(productPriceList);   // displaying data
30.     }
31. }
32. [25000.0, 28000.0, 28000.0]
33. [25000.0, 28000.0, 28000.0]
34. [25000.0, 28000.0, 28000.0]
35. [25000.
```

Major features of Java 7 include:

- Language enhancements grouped under a Project Coin
- String object in switch statement
- Multiple exceptions handling to eliminate duplication of codes
- Upgraded class-loader architecture
- Improved type interference for generic instance
- Library support for ECC (elliptic curve cryptography) algorithms
- Upgraded Rowset 1.1 and JDBC 4.1
- Improved Managed Beans
- Automatic resource management in try-statement
- Concurrency and collections updates
- Compressed 64-bit pointers

- **Major features of Java 8 include:**

Language-level support for Lambda Expressions

- Interface default and Static Methods
- Unsigned Integer Arithmetic
- Concurrent API enhancements
- New Date and Time API
- Parallel Sorting
- Null Reference Template
- New JavaScript Engine, Nashorn
- New and improved Stream API

Java 7	Java 8
Java SE 7 was codenamed Dolphin.	Code name for Java SE 8 is Spider.
Java 7 is supported on Win XP.	Java 8 is not officially supported on Win XP.
Java 7 brings JVM support for dynamically-typed languages plus Type Interference for Generic Instance creation.	Java 8 brings the most anticipated feature for the programming language called Lambda Expressions, a new language feature which allows users to code local functions as method arguments.
Users can catch multiple exception types in one catch block which could be impossible before JDK 7.	Java 8 brings its own new specialized API for Date and Time manipulation.
Small language enhancements was brought to simplify common programming tasks such as automatic resource management, string object in switch, better exception handling, etc.	New and improved JavaScript engine, Nashorn which allows developers to run the script on a JVM. The idea was to implement a lightweight JavaScript runtime in the programming language with a native JVM.

Java 8 Features

Oracle released a new version of Java as Java 8 in March 18, 2014. It was a revolutionary release of the Java for software development platform. It includes various upgrades to the Java programming, JVM, Tools and libraries.

Java 8 Programming Language Enhancements

Java 8 provides following features for Java Programming:

- Lambda expressions,
- Method references,
- Functional interfaces,
- Stream API,
- Default methods,
- Base64 Encode Decode,
- Static methods in interface,
- Optional class,
- Collectors class,
- ForEach() method,
- Nashorn JavaScript Engine,
- Parallel Array Sorting,
- Type and Repating Annotations,
- IO Enhancements,
- Concurrency Enhancements,
- JDBC Enhancements etc.

Lambda Expressions

Lambda expression helps us to write our code in functional style. It provides a clear and concise way to implement SAM interface(Single Abstract Method) by using an expression. It is very useful in collection library in which it helps to iterate, filter and extract data.

For more information and examples: click here

Method References

Java 8 Method reference is used to refer method of functional interface . It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference.

For more information and examples: click here

Functional Interface

An Interface that contains only one abstract method is known as functional interface. It can have any number of default and static methods. It can also declare methods of object class.

Functional interfaces are also known as Single Abstract Method Interfaces (SAM Interfaces).

For more information and examples: click here

Optional

Java introduced a new class Optional in Java 8. It is a public final class which is used to deal with NullPointerException in Java application. We must import *java.util* package to use this class. It provides methods to check the presence of value for particular variable.

For more information and examples: click here

---

forEach

Java provides a new method forEach() to iterate the elements. It is defined in Iterable and Stream interfaces.

It is a default method defined in the Iterable interface. Collection classes which extends Iterable interface can use forEach() method to iterate elements.

This method takes a single parameter which is a functional interface. So, you can pass lambda expression as an argument.

For more information and examples: click here

Date/Time API

Java has introduced a new Date and Time API since Java 8. The java.time package contains Java 8 Date and Time classes.

For more information and examples: click here

Default Methods

Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default keyword are known as default methods. These methods are non-abstract methods and can have method body.

For more information and examples: click here

Nashorn JavaScript Engine

Nashorn is a JavaScript engine. It is used to execute JavaScript code dynamically at JVM (Java Virtual Machine). Java provides a command-line tool **jjs** which is used to execute JavaScript code.

You can execute JavaScript code by two ways:

1. Using jjs command-line tool, and
2. By embedding into Java source code.

For more information and examples: click here

StringJoiner

Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by a delimiter. Now, you can create string by passing delimiters like comma(,), hyphen(-) etc.

For more information and examples: click here

Collectors

Collectors is a final class that extends Object class. It provides reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria etc.

For more information and examples: click here

Stream API

Java 8 java.util.stream package consists of classes, interfaces and an enum to allow functional-style operations on the elements. It performs lazy computation. So, it executes only when it requires.

For more information and examples: click here

Stream Filter

Java stream provides a method filter() to filter stream elements on the basis of given predicate. Suppose, you want to get only even elements of your list, you can do this easily with the help of filter() method.

This method takes predicate as an argument and returns a stream of resulted elements.

For more information and examples: click here

Java Base64 Encoding and Decoding

Java provides a class Base64 to deal with encryption and decryption. You need to import java.util.Base64 class in your source file to use its methods.

This class provides three different encoders and decoders to encrypt information at each level.

For more information and examples: click here

Java Parallel Array Sorting

Java provides a new additional feature in Arrays class which is used to sort array elements parallelly. The parallelSort() method has added to java.util.Arrays class that uses the JSR 166 Fork/Join parallelism common pool to provide sorting of arrays. It is an overloaded method.

Java 8 Security Enhancements

1) The Java Secure Socket Extension(JSSE) provider enables the protocols Transport Layer Security (TLS) 1.1 and TLS 1.2 by default on the client side.

2) A improved method AccessController.doPrivileged has been added which enables code to assert a subset of its privileges, without preventing the full traversal of the stack to check for other permissions.

3) Advanced Encryption Standard (AES) and Password-Based Encryption (PBE) algorithms, such as PBEWithSHA256AndAES_128 and PBEWithSHA512AndAES_256 has been added to the SunJCE provider.

4) Java Secure Socket Extension (SunJSSE) has enabled Server Name Indication (SNI) extension for client applications by default in JDK 7 and JDK 8 supports the SNI extension for server applications. The SNI extension is a feature that extends the SSL/TLS protocols to indicate what server name the client is attempting to connect to during handshaking.

Sorting Map directly with `Comparators`.

As we know Map is in order, it is a lot of struggle to get it sorted. Now Map interface added default methods which gives you comparators for different styles like `comparingByKey`, `comparingByValue`.

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

```
    map.put("C", "c");

    map.put("B", "b");

    map.put("Z", "z");

    List<Map.Entry<String, String>> sortedByKey = map.entrySet().stream().sorted(Map.Entry.comparingByKey())

        .collect(Collectors.toList());

    sortedByKey.forEach(System.out::println);
```

output :

```
B=b

C=c

Z=z
```

Iterate over map easily with `forEach`.

If you observe the above example code, while printing I used the method `forEach` method. This is very revealing feature so far in Map. We all know how ugly is the old fashioned way of iterating and finally implementers added a default method `forEach`. Now it is super easy to iterate over map just like `List`.

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

    map.put("C", "c");

    map.put("B", "b");

    map.put("Z", "z");

    map.forEach((k, v) -> System.out.println("Key : " + k + " Value : " + v));



output :



Key : B Value : b

Key : C Value : c
```

```
Key : Z Value : z
```

Get rid off ugly if-else condition, use `getOrDefault` method.

Legacy code for checking `containsKey` got moved to default method `getOrDefault`. This method returns the value to which the specified key is mapped, otherwise returns the given defaultValue if this map contains no mapping for the key.

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

  map.put("C", "c");

  String val = map.getOrDefault("B", "Nah!");

  System.out.println(val); // prints Nah!
```

Replace and Remove utilities.

New utility default methods have been added now. `replaceAll` Can replace all the values in a single attempt

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

  map.put("C", "c");
```

```
map.put("B", "b");

map.replaceAll((k, v) -> "x"); // values is "x" for all keys.
```

And `replace(K key, V oldValue, V newValue)` method replaces the entry for the specified key only if currently mapped to the specified value. In the same way you can use `replace`, `remove` methods to check by key and values pairs together.

Do not override keys accidentally use `putIfAbsent`

As the method name is self explanatory, here is an example.

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

map.put("C", "c");

map.put("B", "b");

map.putIfAbsent("B", "x");

System.out.println(map.get("B")); // prints "b"
```

operate directly on values.

Gone are the days when you needed to get the value for specific keys, process it and put them back. Now you can directly modify with help of compute method.

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

map.put("C", "c");

map.put("B", "b");

map.compute("B", (k, v) -> v.concat(" - new "));

System.out.println(map.get("B")); // prints "b - new"
```

Conditional computes are also available. Look at `computeIfPresent`, `computeIfAbsent` methods.

To merge maps use `merge` method.

This is little tricky and more useful when you are combining maps or appending values for duplicated keys.

Docs says

If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. Otherwise, replaces the associated value with the results of the given remapping function, or removes if the result is null.

To demonstrate it simply, just merge the values for a key with old and new, see the below example.

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

```
  map.put("C", "c");


  map.put("B", "b");


  map.merge("B", "NEW", (v1, v2) -> v1 + v2);


  System.out.println(map.get("B")); // prints bNEW
```

How to sort employee data by using employee in stream api?

```
import java.util.ArrayList;

import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class StreamListSorting {
    public static void main(String[] args) {

        // sort employee by salary in ascending order
        List < Employee > employees = new ArrayList < Employee > ();
        employees.add(new Employee(10, "Ramesh", 30, 400000));
        employees.add(new Employee(20, "John", 29, 350000));
        employees.add(new Employee(30, "Tom", 30, 450000));
        employees.add(new Employee(40, "Pramod", 29, 500000));

        List < Employee > employeesSortedList1 = employees.stream()
            .sorted((o1, o2) -> (int)(o1.getSalary() - o2.getSalary())).collect(Collectors.toList());
        System.out.println(employeesSortedList1);

        List < Employee > employeesSortedList2 = employees.stream()
            .sorted(Comparator.comparingLong(Employee::getSalary)).collect(Collectors.toList()); //ascending order
        System.out.println(employeesSortedList2);
    }
}
```

Output:

```
[Employee [id=20, name=John, age=29, salary=350000], Employee [id=10, name=Ramesh, age=30, salary=400000], Employee [id=30,
name=Tom, age=30, salary=450000], Employee [id=40, name=Pramod, age=29, salary=500000]]
[Employee [id=20, name=John, age=29, salary=350000], Employee [id=10, name=Ramesh, age=30, salary=400000], Employ
```

Diff bet functional interface and normal interface?

A regular interface can be created with any number of methods whereas a functional interface can only have one. Regular interfaces have been around since Java one and can be implemented by regular concrete classes. Functional interfaces, introduced in Java 8, are meant to be used with lambdas. Since lambdas are inlined bits of code, and since Java is a strongly and statically type language, lambdas must fit into the Java type system. Functional interfaces are the way in which this is accomplished. Functional interfaces can also have any number of default or static methods, but they can only have one abstract method against which the lambda is defined.

Marker interfaces are a relic of the past and are no longer needed. They were used to categorize or label classes but you would now use annotations to achieve the same goal.

What is the contract between equals() and hashCode() methods in Java?

Every Java object has two very important methods **equals()** and **hashCode()** and these methods are designed to be overridden according to their specific **general contract**. An **Object** class is the parent class of every class, the default implementation of these two methods is already present in each class. However, we can override these methods based on the requirement.

hashCode() Method

```
public int hashCode()
```

This method returns an **integer** value, which is referred to as the hash code value of an object. Every Object, at the time of creation assigned with a unique 32-bit, signed int value. This value is the hash code value of that object.

General contract associated with hashCode() method

- The **hashCode()** method should return the same integer value for the same object for each calling of this method unless the value stored in the object is modified.
- If two objects are equal(according to **equals()** method) then the **hashCode()** method should return the same integer value for both the objects.
- But, it is not necessary that the **hashCode()** method will return the distinct result for the objects that are not equal (according to **equals()** method).

what is hashcode and equal method work?

Equals() and Hashcode() in Java

The equals() and hashcode() are the two important methods provided by the **Object** class for comparing objects. Since the Object class is the parent class for all Java objects, hence all objects inherit the default implementation of these two methods. In this topic, we will see the detailed description of equals() and hashcode() methods, how they are related to each other, and how we can implement these two methods in Java.

Java equals()

- The java equals() is a method of *lang.Object* class, and it is used to compare two objects.
- To compare two objects that whether they are the same, it compares the values of both the object's attributes.
- By default, two objects will be the same only if stored in the same memory location.

Syntax:

1. **public boolean** equals(Object obj)

Parameter:

obj: It takes the reference object as the parameter, with which we need to make the comparison.

Returns:

It returns the true if both the objects are the same, else returns false.

General Contract of equals() method

There are some general principles defined by Java SE that must be followed while implementing the equals() method in Java. The equals() method must be:

- ***reflexive***: An object x must be equal to itself, which means, for object x, **equals(x)** should return true.
- ***symmetric***: for two given objects x and y, *x.equals(y)* must return true if and only if *equals(x)* returns true.
- ***transitive***: for any objects x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- ***consistent***: for any objects x and y, the value of x.equals(y) should change, only if the property in equals() changes.
- For any object x, the *equals(null)* must return false.

Java hashcode()

- A **hashcode** is an integer value associated with every object in Java, facilitating the hashing in hash tables.
- To get this hashcode value for an object, we can use the hashcode() method in Java. It is the means ***hashcode() method that returns the integer hashcode value of the given object***.
- Since this method is defined in the Object class, hence it is inherited by user-defined classes also.
- The hashcode() method returns the same hash value when called on two objects, which are equal according to the equals() method. And if the objects are unequal, it usually returns different hash values.

Syntax:

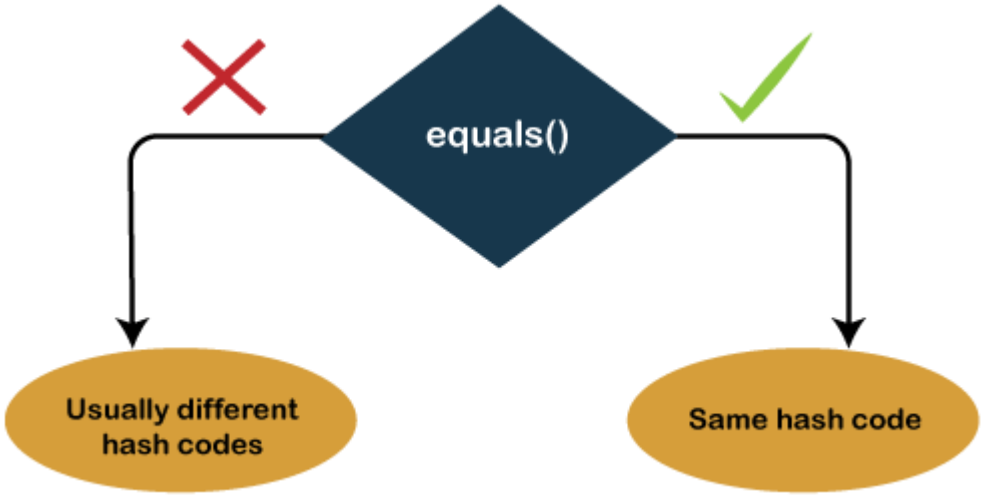
```
1. public int hashCode()
```

Returns:

It returns the hash code value for the given objects.

Contract for hashcode() method in Java

- If two objects are the same as per the equals(Object) method, then if we call the hashCode() method on each of the two objects, it must provide the same integer result.



how does string memory allocated?

Java Strings are immutable objects. In a way **each time you create a String, there will be char[] memory allocated with number of chars in String**. If you do any manipulations on that String it will be brand new object and with the length of chars there will be memory allocation done.

How is memory allocated string?

Strings are stored on the heap area in a separate memory location known as String Constant pool. String constant pool: It is a separate block of memory where all the String variables are held. String str1 = "Hello"; directly, then JVM creates a String object with the given value in a String constant pool.

String Pool in Java

In Java, **String** is the most important topic. There is a number of concepts related to String but the **string pooling concept** is one of them. The **String pooling concept in Java** is a bit tricky. So, in this section, we are going to discuss the **String Pool** or **String Intern concept**.

What is the string pool?

String pool is nothing but a storage area in Java heap where string literals stores. It is also known as **String Intern Pool** or **String Constant Pool**. It is just like object allocation. By default, it is empty and privately maintained by the **Java String** class. Whenever we create a string the string object occupies some space in the heap memory. Creating a number of strings may increase the cost and memory too which may reduce the performance also.

The JVM performs some steps during the initialization of string literals that increase the performance and decrease the memory load. To decrease the number of String objects created in the JVM the String class keeps a pool of strings.

When we create a string literal, the JVM first check that literal in the String pool. If the literal is already present in the pool, it returns a reference to the pooled instance. If the literal is not present in the pool, a new String object takes place in the String pool.

Difference between == and .equals() methods

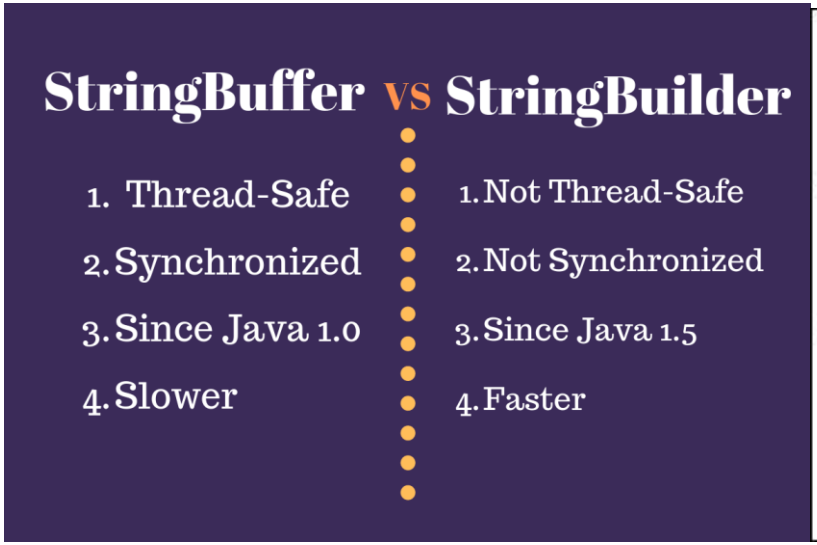
SL No	==	Equals()
1	It will use the memory location for comparison rather that the data. (checks wheather two strings are pointing to same location or not)	I will check the content between two object rather that the memory location. (equals method checks wheather the strings are same or not)
2	It used for comparing two Strings Objects rather than value.	.equels is used for comparing two strings
3	It is used for checking the datatypes values like int float double..	.equals is used to check the Objects like String Vector ....
4	It is used to compare the values of primitive typed variables and the memory location of Objects.	equals is used to compare the content of the two objects.

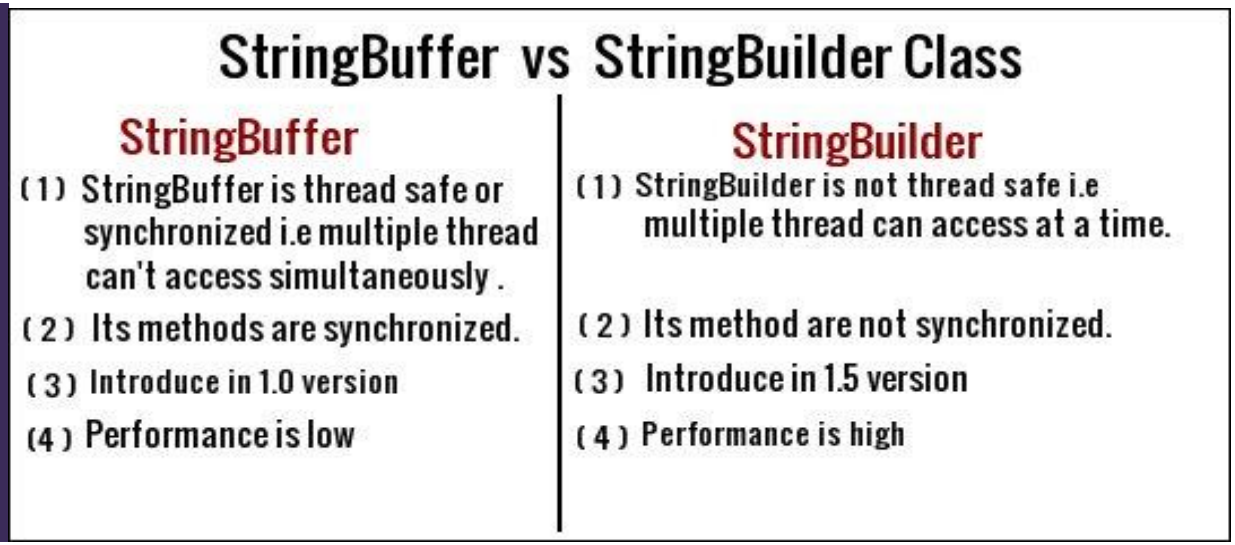
5	`String s1="ashu";`
	`String s2="ashu";`
	
	`//same Object refrence`
	`if(s1==s2)`
	`System.out.println("Hi first time");`
		`String s3 = new String("ashu");`
		`String s4 = new String("ashu");`
		
		`if(s3.equals(s4)) // this will return true as it wil check object contents`
		
		`System.out.println("H!! Third Time");`

memory of string?

Thanks to the immutability of Strings in Java, **the JVM can optimize the amount of memory allocated for them by storing only one copy of each literal String in the pool**. This process is called interning. When we create a String variable and assign a value to it, the JVM searches the pool for a String of equal value.

diff bet stringbuffer and stringbuilder?





how will string do mutable?

In a mutable string, we can **change the value of string and JVM doesn't create a new object**. In a mutable string, we can change the value of the string in the same object. To create a mutable string in java, Java has two classes StringBuffer and StringBuilder where String class is used to the immutable string

write a program to unduplicate character?

```
import java.util.*;

class GFG
{
    static String removeDuplicate(char str[], int n)
    {
        // Used as index in the modified string
        int index = 0;

        // Traverse through all characters
        for (int i = 0; i < n; i++)
        {

            // Check if str[i] is present before it
            int j;
            for (j = 0; j < i; j++)
            {
                if (str[i] == str[j])
                {
                    break;
                }
            }

            // If not present, then add it to
            // result.
            if (j == i)
            {
                str[index++] = str[i];
            }
```

```
        }
        return String.valueOf(Arrays.copyOf(str, index));
    }

    // Driver code
    public static void main(String[] args)
    {
        char str[] = "geeksforgeeks".toCharArray();
        int n = str.length;
        System.out.println(removeDuplicate(str, n));
    }
}
```

write program to reverse a string?

```
1. public class StringFormatter {
2. public static String reverseString(String str){
3.     StringBuilder sb=new StringBuilder(str);
4.     sb.reverse();
5.     return sb.toString();
```

```
1. public class TestStringFormatter {
2. public static void main(String[] args) {
3.     System.out.println(StringFormatter.reverseString("my name is khan"));
4.     System.out.println(StringFormatter.reverseString("I am sonoo jaiswal"));
5.     }
6. }
```

Output:

```
nahk si eman ym
lawsiaj oonos ma I
```

write a program for string palindrome

```
1.  class PalindromeExample{
2.   public static void main(String args[]){
3.    int r,sum=0,temp;
4.    int n=454;//It is the number variable to be checked for palindrome
5.
6.    temp=n;
7.    while(n>0){
8.     r=n%10;  //getting remainder
9.     sum=(sum*10)+r;
10.    n=n/10;
11.   }
12.   if(temp==sum)
13.    System.out.println("palindrome number ");
14.   else
15.    System.out.println("not palindrome");
16. }
17. }
```

Output:

palindrome number

PalindromeCheck.java

```java
package com.palindrome.string;
import java.util.Scanner;

public class PalindromeCheck {


 public static void main(String[] args) {

  Scanner scanner = new Scanner(System.in);
  System.out.print("Enter string : ");

  String str = scanner.nextLine();
  String reverseStr = "";

  for(int i = str.length() - 1; i >= 0; i--){
   reverseStr = reverseStr + str.charAt(i);
  }

  if(str.equals(reverseStr)){
   System.out.println("String is palindrome");
  } else {
   System.out.println("String is not palindrome");
  }
 }
}
```

Problems Javadoc Declaration Console

```
<terminated> PalindromeCheck [Java Application] C:\Program Files\Java\jre1
Enter string : abba
String is palindrome
```

What is the root class in Java?

The Object class of the java.lang package is the root class in Java i.e. It is the super class of every user-defined/predefined class n Java. All objects, including arrays, implement the methods of this class.

The reason for this is to have common functionalities such as synchronization, garbage collection, collection support, object cloning for all objects in Java.

The class named Class of java.lang package provides a method named getSuperclass() this method returns the Class representing the superclass of the current Class.

So, Create a sample concrete class and try to get the name of its super class using this method.

Example

```
public class Sample {
  public static void main(String args[]) {
    Sample obj = new Sample ();
    Class cls = obj.getClass().getSuperclass();
    System.out.println(cls.getName());
  }
}
```

Output

```
java.lang.Object
```

How many ways to create object?

In Java, we can create objects with 6 different methods which are:

1. By new keyword.
2. By newInstance() method of Class class.
3. By newInstance() method of constructor class.
4. By clone() method.

5. By deserialization.
6. By factory method

How many methods in object class?
There are **11** methods in Object class

Method	Description
public final Class getClass()	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.
public final void notify()	wakes up single thread, waiting on this object's monitor.
public final void notifyAll()	wakes up all the threads, waiting on this object's monitor.
public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

protected void finalize()throws Throwable	is invoked by the garbage collector before object is being garbage collected.

What is Hibernate entity class?

Entity: In general entity is an object that has some distinct value. In a persistent storage mechanism, an entity is a business object. Each entity has an associated table in relational database. Each instance of the entity represents a row of the table. Entity class is **an annotated POJO (Plain Old java Object)**.

The primary key class must fulfill several conditions:

1. It should be defined using @EmbeddedId or @IdClass annotations.
2. It should be public, serializable and have a public no-arg constructor.
3. Finally, it should implement equals() and hashCode() methods

What is composite primary key in Hibernate?

A composite primary key, also called a composite key, is **a combination of two or more columns to form a primary key for a table**.
...
2. Composite Primary Keys

1. The composite primary key class must be public.
2. It must have a no-arg constructor.

3. It must define the equals() and hashCode() methods.
4. It must be Serializable.

Difference between lazy and eager loading in Hibernate

The first thing that we should discuss here is what lazy loading and eager loading are: Eager Loading is a design pattern in which data initialization occurs on the spot. Lazy Loading is a design pattern that we use to defer initialization of an object as long as it's possible.

Lazy and Eager are two types of data loading strategies in ORMs such as hibernate and eclipse Link. These data loading strategies we used when one entity class is having references to other Entities like Employee and Phone (phone in the employee).

Lazy Loading − Associated data loads only when we explicitly call getter or size method.

- Use Lazy Loading when you are using one-to-many collections.
- Use Lazy Loading when you are sure that you are not using related entities.

Egare Loading − Data loading happens at the time of their parent is fetched.

- Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.
- Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.

Sr. No.	Key	Lazy	Eager
1	Fetching	In Lazy loading, associated data loads only	In Eager loading, data loading happens

Sr. No.	Key	Lazy	Eager
	strategy	when we explicitly call getter or size method.	at the time of their parent is fetched
2	Default Strategy in ORM Layers	ManyToMany and OneToMany associations used lazy loading strategy by default.	ManyToOne and OneToOne associations used lazy loading strategy by default.
3	Loading Configuration	It can be enabled by using the annotation parameter : fetch = FetchType.LAZY	It can be enabled by using the annotation parameter : fetch = FetchType.EAGER
4	Performance	Initial load time much smaller than Eager loading	Loading too much unnecessary data might impact performance

Hibernate merge() vs. update()

In Hibernate, both update() and merge() methods are used to convert the detached state object into the persistent state.

session.merge()

The **merge()** method is used when we want to change a **detached entity** into the **persistent state** again, and it will automatically update the database. The main aim of the merge() method is to update the changes in the database made by the persistent object. The merge() method is provided by the **Session** interface and is available in **org.hibernate.session** package.
There are two merge() methods available in the Session interface with different parameters:

- merge(Object object)

- merge(String entityName, Object object)

Both of the above methods perform the same functions.

How to use session.merge()

```
Employee employee = session.merge(emp);
```

session.update()

The **update()** method is used to save the object in the database. We can use the update() method when the Session does not contain any persistent object with the same primary key. If the Session has a persistent object with the same primary key, it will throw an exception.
This method adds an entity object to the persistent state, and more changes are tracked and saved when the transaction is committed. The update() method does not return anything. The update() method is provided by the **Session** interface and is available in **org.hibernate.session** package.
There are two update() methods available in the Session interface with different parameters:

- update(Object object)
- update(String entityName, Object object)

Both of the above methods perform the same functions.

How to use session.update()

```
Employee employee = session.update(emp);
```

Difference between merge() and update

Parameter	merge()	update()
Usage	A merge() method is used to update the database. It will also update the database if the object already exists.	An update() method only saves the data in the database. If the object already exists, no update is performed.

Exception	It will update the object in the database without any exception.	When creating a detached object into persistent state, it will throw an exception only when the Session contains a persistent object with the same primary key.

Difference Between get() and load() in Hibernate

In hibernate, get() and load() are two methods which is used to fetch data for the given identifier. They both belong to Hibernate session class. Get() method return null, If no row is available in the session cache or the database for the given identifier whereas load() method throws object not found exception.

Sr. No.	Key	Get()	Load()
1	Basic	It is used to fetch data from the database for the given identifier	It is also used to fetch data from the database for the given identifier
2	Null Object	It object not found for the given identifier then it will return null object	It will throw object not found exception
3	Lazy or Eager loading	It returns fully initialized object so this method eager load the object	It always returns proxy object so this method is lazy load the object
4	Performance	It is slower than load() because it return fully initialized object which impact the performance	It is slightly faster.

Sr. No.	Key	Get()	Load()
		of the application	
5.	Use Case	If you are not sure that object exist then use get() method	If you are sure that object exist then use load() method

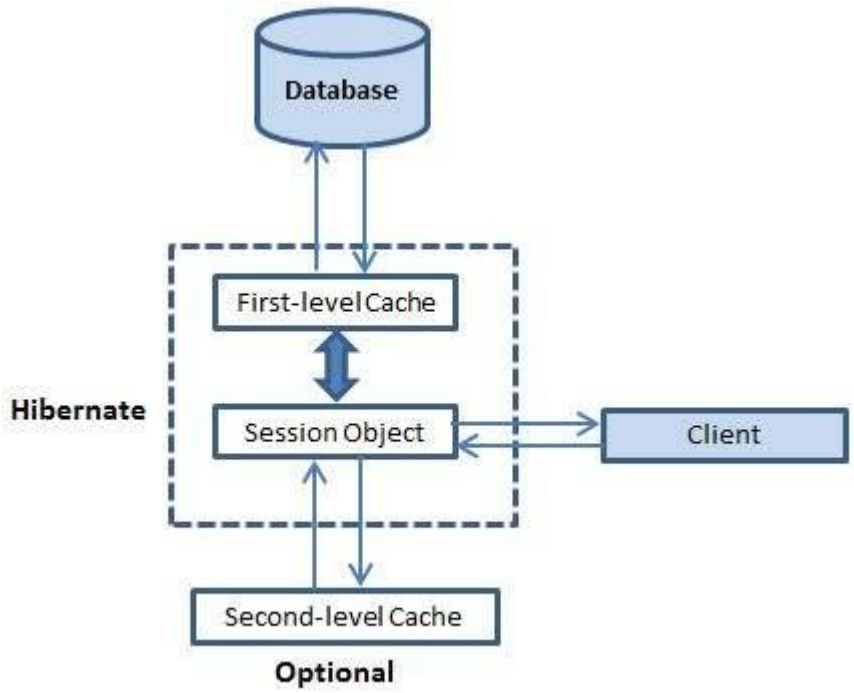
Hibernate - Caching

Previous Page

Next Page

Caching is a mechanism to enhance the performance of a system. It is a buffer memorythat lies between the application and the database. Cache memory stores recently used data items in order to reduce the number of database hits as much as possible.

Caching is important to Hibernate as well. It utilizes a multilevel caching scheme as explained below −



First-level Cache

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass. The Session object keeps an object under its own power before committing it to the database.

If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.

Second-level Cache

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.

Any third-party cache can be used with Hibernate. An **org.hibernate.cache.CacheProvider** interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

Query-level Cache

Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.

This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.

The Second Level Cache

Hibernate uses first-level cache by default and you have nothing to do to use first-level cache. Let's go straight to the optional second-level cache. Not all classes benefit from caching, so it's important to be able to disable the second-level cache.

The Hibernate second-level cache is set up in two steps. First, you have to decide which concurrency strategy to use. After that, you configure cache expiration and physical cache attributes using the cache provider.

Concurrency Strategies

A concurrency strategy is a mediator, which is responsible for storing items of data in the cache and retrieving them from the cache. If you are going to enable a second-level cache, you will have to decide, for each persistent class and collection, which cache concurrency strategy to use.

- **Transactional** − Use this strategy for read-mostly data where it is critical to prevent stale data in concurrent transactions, in the rare case of an update.
- **Read-write** − Again use this strategy for read-mostly data where it is critical to prevent stale data in concurrent transactions, in the rare case of an update.
- **Nonstrict-read-write** − This strategy makes no guarantee of consistency between the cache and the database. Use this strategy if data hardly ever changes and a small likelihood of stale data is not of critical concern.
- **Read-only** − A concurrency strategy suitable for data, which never changes. Use it for reference data only.

Hibernate - Sessions

Previous Page

Next Page

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. The main function of the Session is to offer, create, read, and delete operations for instances of mapped entity classes.

Instances may exist in one of the following three states at a given point in time −

- **transient** − A new instance of a persistent class, which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
- **persistent** − You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
- **detached** − Once we close the Hibernate Session, the persistent instance will become a detached instance.

A Session instance is serializable if its persistent classes are serializable. A typical transaction should use the following idiom −

```
Session session = factory.openSession();
Transaction tx = null;

try {
  tx = session.beginTransaction();
  // do some work
  ...
  tx.commit();
}

catch (Exception e) {
  if (tx!=null) tx.rollback();
  e.printStackTrace();
} finally {
  session.close();
}
```

If the Session throws an exception, the transaction must be rolled back and the session must be discarded.

Session Interface Methods

There are number of methods provided by the **Session** interface, but I'm going to list down a few important methods only, which we will use in this tutorial. You can check Hibernate documentation for a complete list of methods associated with **Session** and **SessionFactory**.

code to configure database in hibernate?

```
if (sessionFactory == null) {
    try {
        Configuration configuration = new Configuration();

        // Hibernate settings equivalent to hibernate.cfg.xml's properties
        Properties settings = new Properties();
        settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
        settings.put(Environment.URL, "jdbc:mysql://localhost:3306/hibernate_db?useSSL=false");
        settings.put(Environment.USER, "root");
        settings.put(Environment.PASS, "root");
        settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");

        settings.put(Environment.SHOW_SQL, "true");

        settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

        settings.put(Environment.HBM2DDL_AUTO, "create-drop");

        configuration.setProperties(settings);

        configuration.addAnnotatedClass(Student.class);

        ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();

        sessionFactory = configuration.buildSessionFactory(serviceRegistry);
    } catch (Exception e) {
```

add JPA entity mapping class

What is Hibernate? List the advantages of hibernate over JDBC.

- Hibernate is used convert object data in JAVA to relational database tables.
- It is an open source Object-Relational Mapping (ORM) for Java.
- Hibernate is responsible for making data persistent by storing it in a database.

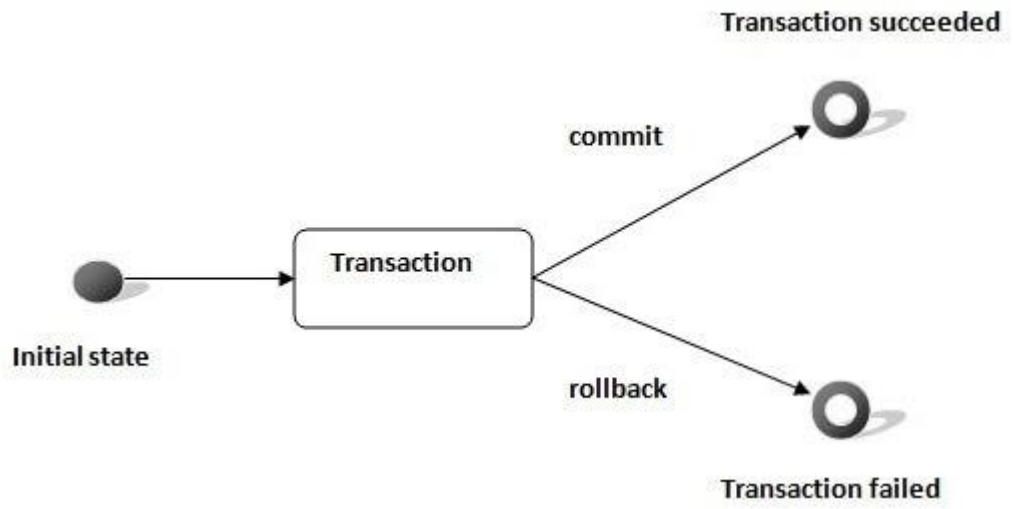
JDBC	Hibernate
JDBC maps Java classes to database tables (and from Java data types to SQL data types)	Hibernate automatically generates the queries.
With JDBC, developer has to write code to map an object model's data to a relational data model.	Hibernate is flexible and powerful ORM to map Java classes to database tables.
With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java. So with JDBC, mapping between Java objects and database tables is done manually.	Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
Require JDBC Driver for different types of database.	Makes an application portable to all SQL databases.
Handles all create-read-update-delete (CRUD) operations using SQL Queries.	Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
Working with both Object-Oriented software and Relational Database is complicated task with JDBC.	Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.
JDBC supports only native Structured Query Language (SQL)	Hibernate provides a powerful query language Hibernate Query Language-HQL (independent from type of database)

List the advantages of hibernate over JDBC

- Hibernate is flexible and powerful ORM to map Java classes to database tables.
- Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects, hence reducing the development time and maintenance cost.
- Hibernate automatically generates the queries.
- It makes an application portable to all SQL databases.
- Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
- Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

Hibernate Transaction?

A **transaction** simply represents a unit of work. In such case, if one step fails, the whole transaction fails (which is termed as atomicity). A transaction can be described by ACID properties (Atomicity, Consistency, Isolation and Durability).



Hibernate Inheritance Mapping?

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

Table Per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table .

Table Per Hierarchy using xml file
Table Per Hierarchy using Annotation

Table Per Concrete class

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

Table Per Concrete class using xml file
Table Per Concrete class using Annotation

Table Per Subclass

In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

TablePerSubclassusingxmlfile
Table Per Subclass using Annotation

how to create a proxy object of hibernate?

By definition, a proxy is “a function authorized to act as the deputy or substitute for another”. This applies to Hibernate when we **call Session. load()** to create what is called an uninitialized proxy of our desired entity class. This subclass will be the one to be returned instead of querying the database directly.

without hitting the database how to get data from database?

When we call session.get() method hibernate will hit the database and returns the original object.

When you call session.load() method, it will always return a “proxy” object without querying the database.

My question is if load() method is not hitting the database then how it know whether the requested information is available in the database.

Why do we need Hibernate framework?

Hibernate's primary feature is **mapping from Java classes to database tables, and mapping from Java data types to SQL data types**. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

What is the difference between DriverManager and DataSource?

DataSource and the DriverManager are the two basic ways to connect to a database. **The DriverManager is older facility, DataSource is newer**. It is recommended to use the new DataSource facility to connect to databases and other resources. DataSource facility has several advantages over DriverManager facility.

What is session.begintransaction in hibernate

Transaction Interface in Hibernate

In hibernate framework, we have Transaction interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC). **A transaction is associated with Session and instantiated by calling session.** beginTransaction().

hibernate mappings are one of the key features of hibernate . they **establish the relationship between two database tables as attributes in your model**. that allows you to easily navigate the associations in your model and criteria queries

What is mapping explain different types of mapping in hibernate?

The types declared and used in the mapping files are not Java data types; they are not SQL database types either. These types are called Hibernate mapping types, which **can translate from Java to SQL data types and vice versa**.

Throw vs Throws

Throw	Throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

The try-with-resources statement

In Java, the try-with-resources statement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program. The try-with-resources statement ensures that each resource is closed at the end of the statement execution.

You can pass any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable.

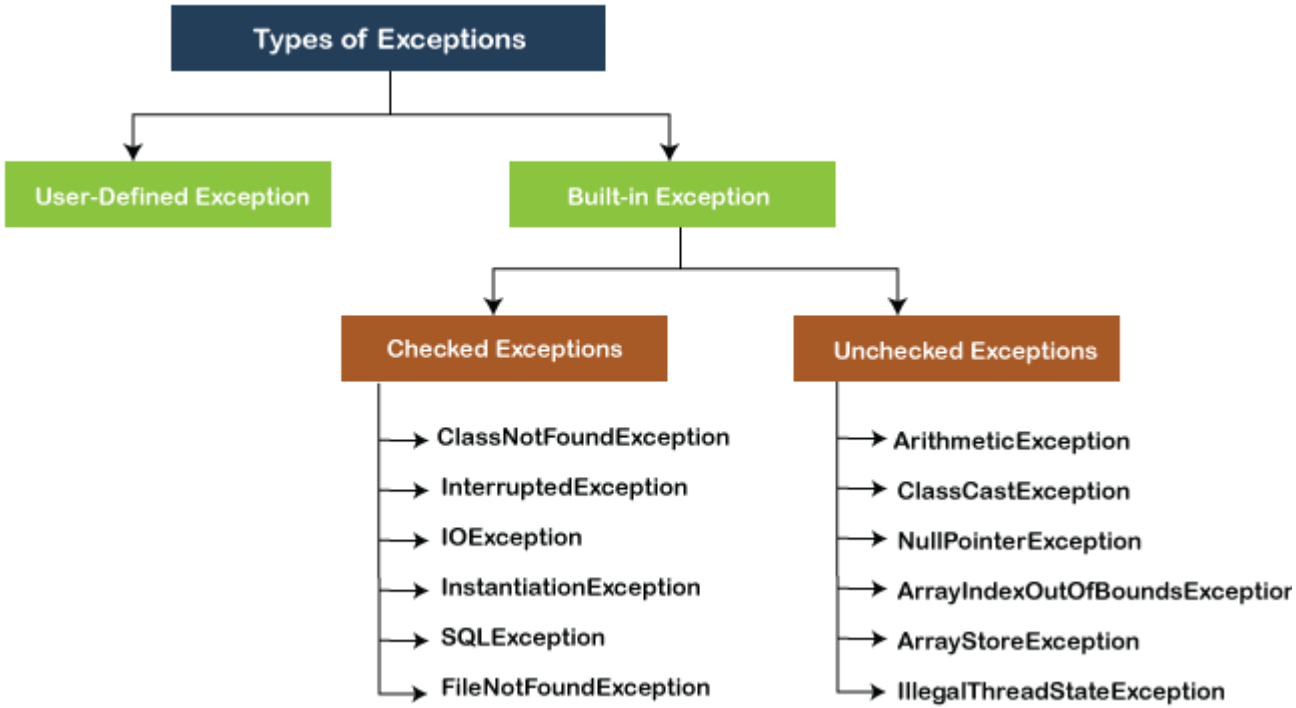
Java Try with Resource

```
private static void readFile() throws IOException {
try(FileInputStream input = new FileInputStream("file.txt")) {
                private static void readFile() throws IOException {
                try(  FileInputStream input = new FileInputStream("file.txt");
                BufferedInputStream bufferedInput = new BufferedInputStream(input)
                ) {
public interface AutoCloseable {
public void close() throws Exception;
}
```

How do we create custom exception in Java?

Here are the steps:

1. Create a new class whose name should end with Exception like ClassNameException. ...
2. Make the class extends one of the exceptions which are subtypes of the java. ...
3. Create a constructor with a String parameter which is the detail message of the exception.



To serialize an object means **to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object**. A Java object is serializable if its class or any of its superclasses implements either the java. io. Serializable interface or its subinterface, java.

Volatile keyword is **used to modify the value of a variable by different threads**. It is also used to make classes thread safe. It means that multiple threads can use a method and instance of the classes at the same time without any problem.

transient is **a variables modifier used in serialization**. At the time of serialization, if we don't want to save value of a particular variable in a file, then we use transient keyword. When JVM comes across transient keyword, it ignores original value of the variable and save default value of that variable data type.

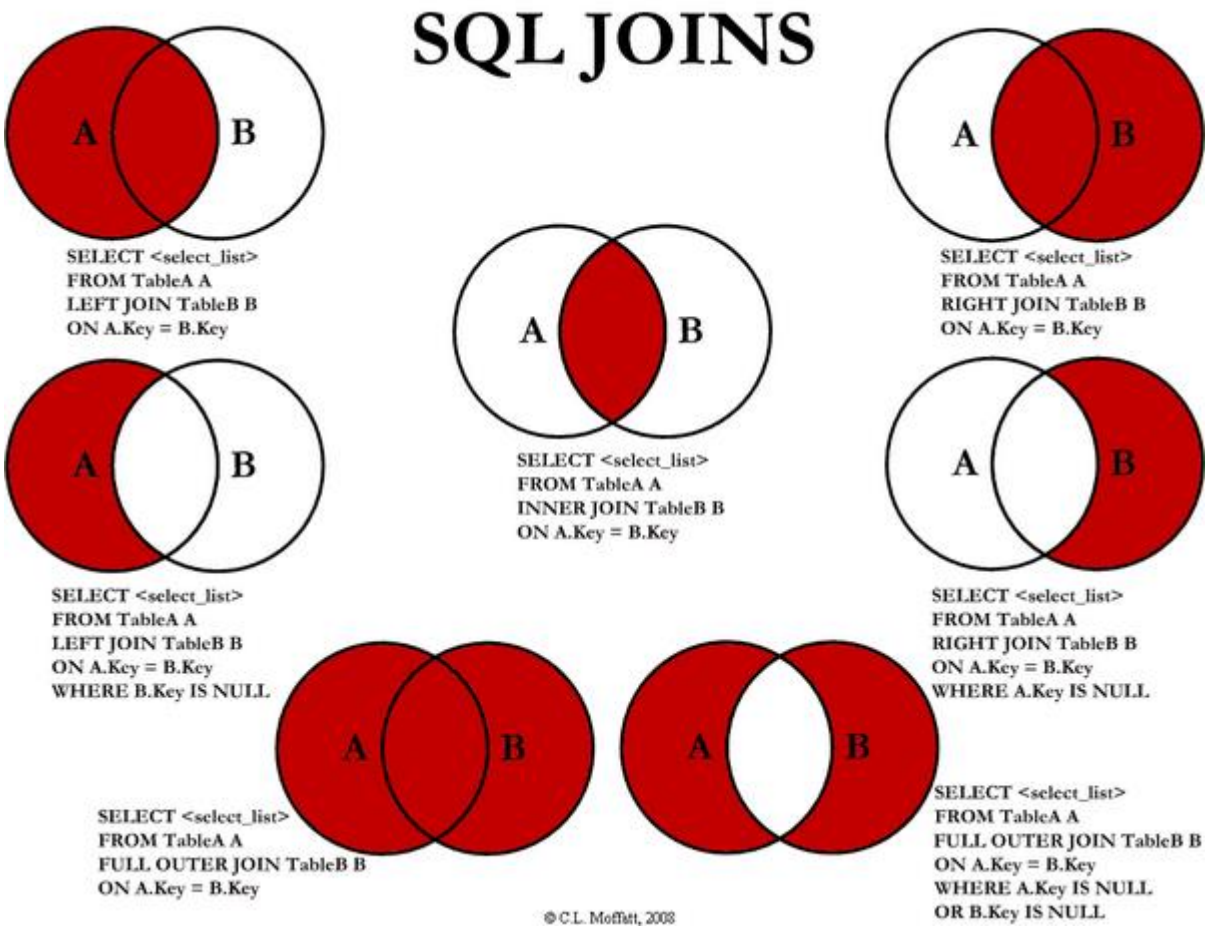
Synchronization in java is **the capability to control the access of multiple threads to any shared resource**. In the Multithreading concept, multiple threads try to access the shared resources at a time to produce inconsistent results. The synchronization is necessary for reliable communication between threads.

INNER JOIN gets all records that are common between both tables based on the supplied ON clause.

LEFT JOIN gets all records from the LEFT linked and the related record from the right table ,but if you have selected some columns from the RIGHT table, if there is no related records, these columns will contain NULL.

RIGHT JOIN is like the above but gets all records in the RIGHT table.

FULL JOIN gets all records from both tables and puts NULL in the columns where related records do not exist in the opposite table.



write query to findout second highest salary of an employee?

Second Highest Salary in MySQL without LIMIT

Here is a generic SQL query to find the second highest salary, which will also work fine in MySQL. This solution uses a subquery to first exclude the maximum salary from the data set and then again finds the maximum salary, which is effectively the second maximum salary from the Employee table.

```
SELECT MAX(salary) FROM Employee
WHERE Salary NOT IN ( SELECT Max(Salary) FROM Employee);
```

what is inner join and outer join?

(INNER) JOIN : **Returns records that have matching values in both tables**. LEFT (OUTER) JOIN : Returns all records from the left table, and the matched records from the right table. RIGHT (OUTER) JOIN : Returns all records from the right table, and the matched records from the left table.

write a query to copy one table into another table?

The SQL INSERT INTO SELECT Statement

The `INSERT INTO SELECT` statement copies data from one table and inserts it into another table.

The `INSERT INTO SELECT` statement requires that the data types in source and target tables match.

Note: The existing records in the target table are unaffected.

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
WHERE condition;
```

Diff bet drop delete and truncate?

TRUNCATE	DELETE
Truncate is used to delete the all records in table	Delete is used to delete the row level and table level data
We can't rollback the data	We can rollback data if we maintain the transaction
It's auto committed	It's explicit committed
It's a DDL (Data Definition Language) Command	It's a DML (Data Manipulation Language) Command
It's faster than the delete	It's very slow when we compare with TRUNCATE

What is Normalization?

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.
The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

QL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

Try it Yourself »

Cursor in SQL Server

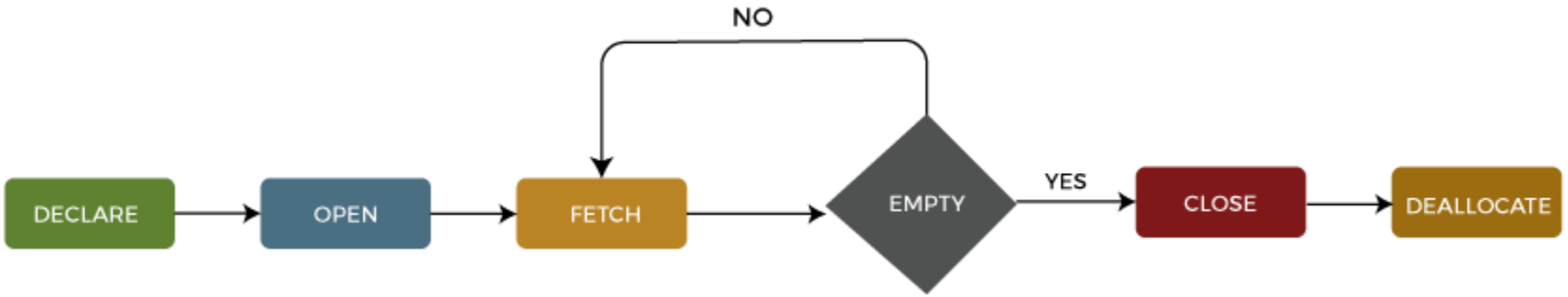
A cursor in SQL Server is a d**atabase object that allows us to retrieve each row at a time and manipulate its data**. A cursor is nothing more than a pointer to a row. It's always used in conjunction with a SELECT statement. It is usually a collection of SQL logic that loops through a predetermined number of rows one by one. A simple illustration of the cursor is when we have an extensive database of worker's records and want to calculate each worker's salary after deducting taxes and leaves.

The SQL Server **cursor's purpose is to update the data row by row, change it, or perform calculations that are not possible when we retrieve all records at once**. It's also useful for performing administrative tasks like SQL Server database backups in sequential order. Cursors are mainly used in the development, DBA, and ETL processes.

This article explains everything about SQL Server cursor, such as cursor life cycle, why and when the cursor is used, how to implement cursors, its limitations, and how we can replace a cursor.

Life Cycle of the cursor

We can describe the life cycle of a cursor into the **five different sections** as follows:



1: Declare Cursor

The first step is to declare the cursor using the below SQL statement:

```
1. DECLARE cursor_name CURSOR
2.     FOR select_statement;
```

The MySQL GROUP BY Statement

The `GROUP BY` statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
GROUP BY column_name(s)
ORDER BY column_name(s);
```

How can get second highest salary in MySQL?

Second Maximum Salary in MySQL using **LIMIT**

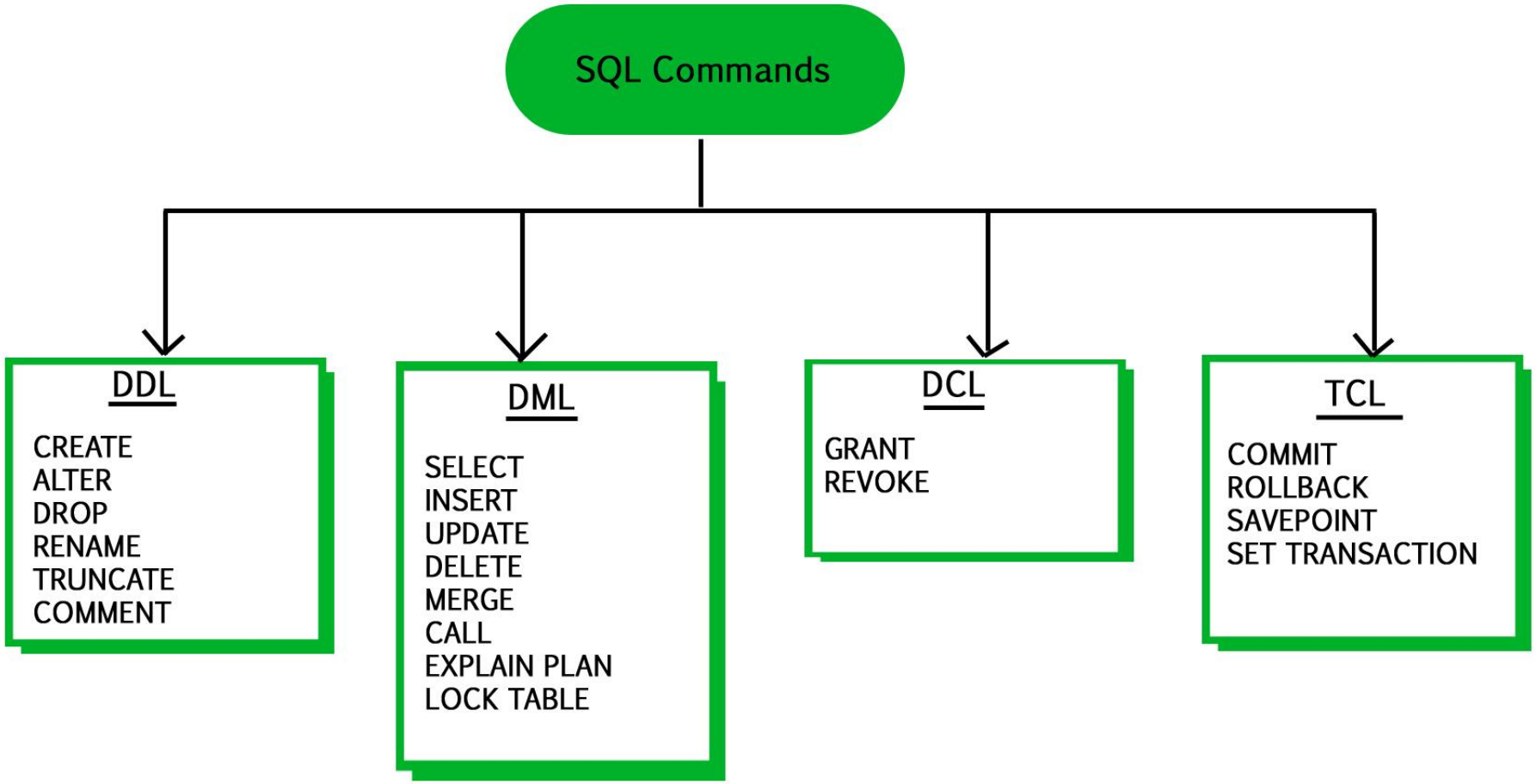
SELECT Salary FROM (SELECT Salary FROM Employee ORDER BY salary DESC LIMIT 2) AS Emp ORDER BY salary LIMIT 1; In this solution, we have first sorted all salaries from the Employee table in decreasing order, so that the 2 highest salaries come at top of the result set.

Basic Differences between Stored Procedure and Function in SQL Server

1. The function must return a value but in **Stored Procedure** it is optional. Even a procedure can return zero or n values.
2. Functions can have only input parameters for it whereas Procedures can have input or output parameters.
3. Functions can be called from Procedure whereas Procedures cannot be called from a Function.

Data Definition Language(DDL) helps you to define the database structure or schema. Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data.

DDL	DML
It is Data Definition Language	It is Data Manipulation Language
These are used to define data structure	It is used to manipulate the existing databases.
It is used to define database structure or schema	It is used for managing data within schema objects
Commands are: CREATE, ALTER, DROP, TRUNCATE, RENAME	Commands are: SELECT, INSERT, DELETE, UPDATE, MERGE, CALL
It works on whole table	It works on one or more rows
It do not have a where clause to filter	It have where clause to filter records
Changes done by DDL commands cannot be rolled back	Changes can be rolled back
It is not further classified.	It is further classified as procedural and non procedural DML's
Example:- drop table tablename;	Select * from employee


SQL Commands
DDL
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
DML
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
DCL
GRANT
REVOKE
TCL
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION