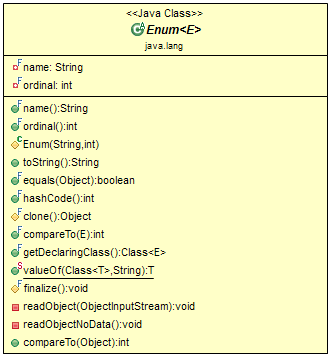
**What Will We Learn?**

1. Enum Basics
2. Define an Enum inside a Class
3. Using Enums in if-else if Statements
4. Using Enums in switch Statements
5. How to Iterate over an Enum?
6. Enum Types Example in Real Time Projects
7. Enum can implements interface and Enum can be nested

**1. Enum Basics**

An enum type is a special data type that enable for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. Common examples include compass directions (values of NORTH, SOUTH, EAST, and WEST) and the days of the week.

Below class diagram shows a list of methods or APIs Enum type provides.

**[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEisk1VJN2rH3SfsSpwBdydLqMSZJ7dpGzaTu6HPYvo3ZkLZY3rsfOT0qYm07-rXEpUefqoBTIxfoqa9NkgopU84OXVZYsfKy5nKOyNAqdiv4AOK1bt7mmGcsK-39RZ6IwDBU61Kacdd6tU/s1600/enum-class-diagram.png)**

**Java Enum Types Key points**

* enum improves type safety
* names of an enum type's fields are in uppercase letters.
* enum can be easily used in switch
* enum can be traversed
* enum can have fields, constructors, and methods
* We cannot create an instance of enum because the constructor of an enum type is private(If you don't declare private compiler internally creates private constructor).
* We can write abstract methods in the enum and we can provide an implementation for the same.
* enum may implement many interfaces but cannot extend any class because it internally extends Enum class In the Java programming language, you define an enum type by using the enum keyword. For example, you would specify a days-of-the-week enum type as:

public enum Day {

SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

THURSDAY, FRIDAY, SATURDAY;

}

Let's write a source code to use the *Day* enum defined above: In real projects, we basically use a switch statement to compare enum values like

public class DayEnumTest {

Day day;

public DayEnumTest(){

}

public DayEnumTest(Day day) {

this.day = day;

}

public void tellItLikeItIs() {

switch (day) {

case MONDAY:

System.out.println("Mondays are bad.");

break;

case FRIDAY:

System.out.println("Fridays are better.");

break;

case SATURDAY: case SUNDAY:

System.out.println("Weekends are best.");

break;

default:

System.out.println("Midweek days are so-so.");

break;

}

}

public void mapEnum(Day day){

switch (day) {

case MONDAY:

System.out.println("This is Monday");

break;

case FRIDAY:

System.out.println("This is Friday");

break;

case SATURDAY: case SUNDAY:

System.out.println("Weekends are best.");

break;

default:

System.out.println("Midweek days");

break;

}

}

public static void main(String[] args) {

final DayEnumTest firstDay = new DayEnumTest(Day.MONDAY);

firstDay.tellItLikeItIs();

final DayEnumTest thirdDay = new DayEnumTest(Day.WEDNESDAY);

thirdDay.tellItLikeItIs();

final DayEnumTest fifthDay = new DayEnumTest(Day.FRIDAY);

fifthDay.tellItLikeItIs();

final DayEnumTest sixthDay = new DayEnumTest(Day.SATURDAY);

sixthDay.tellItLikeItIs();

final DayEnumTest seventhDay = new DayEnumTest(Day.SUNDAY);

seventhDay.tellItLikeItIs();

final DayEnumTest dayEnumTest = new DayEnumTest();

dayEnumTest.mapEnum(Day.MONDAY);

}

}

The output is:

Mondays are bad.

Midweek days are so-so.

Fridays are better.

Weekends are best.

Weekends are best.

This is Monday

**2. Define an Enum inside a Class**

Example: Example to define Enum type inside a class.

public class SimpleEnumExample {

enum Days{

SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

}

}

**3. Using Enums in if-else if Statements**

Example: How to use Enums in if-else if Statements.

public class EnumInIfStatement {

public String enumInIf(Days day) {

if(day == Days.SUNDAY) {

return "Its Sunday :-)";

}else if (day == Days.MONDAY) {

return "Its Monday :\*--(";

}else if (day == Days.TUESDAY) {

return "Its Tuesday :\*-(";

}else if (day == Days.WEDNESDAY) {

return "Its Wednesday :\*(";

}else if (day == Days.THURSDAY) {

return "Its Thursday :)";

}else if (day == Days.FRIDAY) {

return "Its Friday ;-D";

}else {

return "Its Saturday :=D";

}

}

}

**4. Using Enums in switch Statements**

Example: How to use Enums in switch Statements.

public class EnumInSwitchStatement {

public String enumInSwitch(Days day) {

switch(day) {

case SUNDAY:

return "Its Sunday!!";

case MONDAY:

return "Its Monday";

case TUESDAY:

return "Its Tuesday";

case WEDNESDAY:

return "Its Wednesday";

default:

return "Rest of the week....";

}

}

}

**5. How to Iterate over an Enum?**

Example: Example of how to iterate over an Enum.

import java.util.ArrayList;

public class EnumIteration {

enum Days{

SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

}

public ArrayList<String> enumIteration() {

Days[] days = Days.values();

ArrayList<String> stringDay = new ArrayList<String>();

for (Days day : days) {

stringDay.add(day.toString());

}

return stringDay;

}

}

**6. Enum Types Example in Real Time Projects**

Let's say we are using multiple databases in our projects like *MySQL* and *Postgres*.

In order to establish a connection with the database, we need to register *Driver* and these *Driver*names are constant or fixed.

Let's write an Enum to make Driver names predefined:

public enum DbType {

MYSQL("com.mysql.jdbc.Driver"), POSTGRESQL("org.postgresql.Driver");

private final String driverClassName;

private DbType(final String driverClassName) {

this.driverClassName = driverClassName;

}

public String getDriverClassName() {

return driverClassName;

}

}

In real projects, we maintain a status for some entity like *ProjectEntity* and its status can be active or inactive. The enum for same is :

public enum ProjectStatus {

ACTIVE, INACTIVE

}

**7. Enum can implements interface and Enum can be nested**

The complete example to demonstrate Enum can implements interface and Enum can be nested.

Let's create a *StatusType* interface that the enum implements :

public interface StatusType {

/\*\*

\* Get the associated status code.

\*

\* @return the status code.

\*/

public int getStatusCode();

/\*\*

\* Get the class of status code.

\*

\* @return the class of status code.

\*/

public Status.Family getFamily();

/\*\*

\* Get the reason phrase.

\*

\* @return the reason phrase.

\*/

public String getReasonPhrase();

}

Let's create enum *Status* and which implements *StatusType* interface and also contains nested enum.

public enum Status implements StatusType {

OK(200, "OK"),

CREATED(201, "Created"),

ACCEPTED(202, "Accepted"),

NO\_CONTENT(204, "No Content"),

RESET\_CONTENT(205, "Reset Content"),

PARTIAL\_CONTENT(206, "Partial Content"),

MOVED\_PERMANENTLY(301, "Moved Permanently"),

FOUND(302, "Found"),

SEE\_OTHER(303, "See Other"),

NOT\_MODIFIED(304, "Not Modified"),

USE\_PROXY(305, "Use Proxy"),

TEMPORARY\_REDIRECT(307, "Temporary Redirect"),

BAD\_REQUEST(400, "Bad Request"),

UNAUTHORIZED(401, "Unauthorized"),

PAYMENT\_REQUIRED(402, "Payment Required"),

FORBIDDEN(403, "Forbidden"),

NOT\_FOUND(404, "Not Found"),

METHOD\_NOT\_ALLOWED(405, "Method Not Allowed"),

NOT\_ACCEPTABLE(406, "Not Acceptable"),

PROXY\_AUTHENTICATION\_REQUIRED(407, "Proxy Authentication Required"),

REQUEST\_TIMEOUT(408, "Request Timeout"),

CONFLICT(409, "Conflict"),

GONE(410, "Gone"),

LENGTH\_REQUIRED(411, "Length Required"),

PRECONDITION\_FAILED(412, "Precondition Failed"),

REQUEST\_ENTITY\_TOO\_LARGE(413, "Request Entity Too Large"),

REQUEST\_URI\_TOO\_LONG(414, "Request-URI Too Long"),

UNSUPPORTED\_MEDIA\_TYPE(415, "Unsupported Media Type"),

REQUESTED\_RANGE\_NOT\_SATISFIABLE(416, "Requested Range Not Satisfiable"),

EXPECTATION\_FAILED(417, "Expectation Failed"),

INTERNAL\_SERVER\_ERROR(500, "Internal Server Error"),

NOT\_IMPLEMENTED(501, "Not Implemented"),

BAD\_GATEWAY(502, "Bad Gateway"),

SERVICE\_UNAVAILABLE(503, "Service Unavailable"),

GATEWAY\_TIMEOUT(504, "Gateway Timeout"),

HTTP\_VERSION\_NOT\_SUPPORTED(505, "HTTP Version Not Supported");

private final int code;

private final String reason;

private final Family family;

/\*\*

\* An enumeration representing the class of status code. Family is used

\* here since class is overloaded in Java.

\*/

public enum Family {

/\*\*

\* {@code 1xx} HTTP status codes.

\*/

INFORMATIONAL,

/\*\*

\* {@code 2xx} HTTP status codes.

\*/

SUCCESSFUL,

/\*\*

\* {@code 3xx} HTTP status codes.

\*/

REDIRECTION,

/\*\*

\* {@code 4xx} HTTP status codes.

\*/

CLIENT\_ERROR,

/\*\*

\* {@code 5xx} HTTP status codes.

\*/

SERVER\_ERROR,

/\*\*

\* Other, unrecognized HTTP status codes.

\*/

OTHER;

/\*\*

\* Get the response status family for the status code.

\*

\* @param statusCode response status code to get the family for.

\* @return family of the response status code.

\*/

public static Family familyOf(final int statusCode) {

switch (statusCode / 100) {

case 1:

return Family.INFORMATIONAL;

case 2:

return Family.SUCCESSFUL;

case 3:

return Family.REDIRECTION;

case 4:

return Family.CLIENT\_ERROR;

case 5:

return Family.SERVER\_ERROR;

default:

return Family.OTHER;

}

}

}

Status(final int statusCode, final String reasonPhrase) {

this.code = statusCode;

this.reason = reasonPhrase;

this.family = Family.familyOf(statusCode);

}

/\*\*

\* Get the class of status code.

\*

\* @return the class of status code.

\*/

@Override

public Family getFamily() {

return family;

}

/\*\*

\* Get the associated status code.

\*

\* @return the status code.

\*/

@Override

public int getStatusCode() {

return code;

}

/\*\*

\* Get the reason phrase.

\*

\* @return the reason phrase.

\*/

@Override

public String getReasonPhrase() {

return toString();

}

/\*\*

\* Get the reason phrase.

\*

\* @return the reason phrase.

\*/

@Override

public String toString() {

return reason;

}

/\*\*

\* Convert a numerical status code into the corresponding Status.

\*

\* @param statusCode the numerical status code.

\* @return the matching Status or null is no matching Status is defined.

\*/

public static Status fromStatusCode(final int statusCode) {

for (final Status s : Status.values()) {

if (s.code == statusCode) {

return s;

}

}

return null;

}

}