Java enums are a special Java type used to define collections of constants. An enum type is a special kind of Java class. It can contain constants, methods etc.

**Enum Example**

Here is a simple enum definition:

public enum Level {

HIGH,

MEDIUM,

LOW

}

You can refer to the constants in the above enum like this:

Level level = Level.HIGH;

Notice how the level variable is of the type Level which is the enum type from the example above. The level variable can take one of the Level enum constants as value (HIGH, MEDIUM or LOW). In this case level is set to HIGH.

**Enums in if Statements**

Being constants, you will often have to compare a variable pointing to an enum constant against the possible constants in the enum type. Here is how that could look:

Level level = ... //assign some Level constant to it

if( level == Level.HIGH) {

} else if( level == Level.MEDIUM) {

} else if( level == Level.LOW) {

}

This code compares the level variable against each of the possible enum constants in the Level enum.

If one of the enum values occur more often than the others, checking for that value in the first if-statement will result in better performance, as less comparison on average are executed. This is not a big difference though, unless the comparisons are executed a lot.

**Enums in switch Statements**

If your enum types contain a lot constants and you need to check a variable against the values as shown in the previous section, using a switch statement might be a good idea.

You can use enums in switch statements like this:

Level level = ... //assign some Level constant to it

switch (level) {

case HIGH : ...; break;

case MEDIUM : ...; break;

case LOW : ...; break;

}

Replace the ... with the code to execute if the level variable matches the given Level constant value. The code could be a simple Java operation, a method call etc.

**Enum Iteration**

You can obtain an array of all the possible values of an enum type by calling its static values() method. All enum types get a static values() method automatically by the Java compiler. Here is an example:

for (Level level : Level.values()) {

System.out.println(level);

}

Running this code would print out all the enum values. Here is the output:

HIGH

MEDIUM

LOW

Notice how the names of the constants themselves are printed out.

**Enum Fields**

You can add fields to an enum. Thus, each constant value gets these fields. The field values must be supplied to the constructor of the enum when defining the constants. Here is an example:

public enum Level {

HIGH (3), //calls constructor with value 3

MEDIUM(2), //calls constructor with value 2

LOW (1) //calls constructor with value 1

; // semicolon needed when fields / methods follow

private final int levelCode;

public Level(int levelCode) {

this.levelCode = levelCode;

}

}

**Enum Methods**

You can add methods to an enum too. Here is an example:

public enum Level {

HIGH (3), //calls constructor with value 3

MEDIUM(2), //calls constructor with value 2

LOW (1) //calls constructor with value 1

; // semicolon needed when fields / methods follow

private final int levelCode;

Level(int levelCode) {

this.levelCode = levelCode;

}

**public int getLevelCode() {**

**return this.levelCode;**

**}**

}

You call en enum method via a reference to one of the constant values. Here is an example:

Level level = Level.HIGH;

System.out.println(level.getLevelCode());

This code would print out the value 3 which is the value of the levelCode field for the constant HIGH.

You are not restricted to simple getter and setter methods. You can also create methods that make calculations based on the field values of the enum constant. If your fields are not declared final you can even modify the values of the fields (although that may not be so good an idea, considering that the enums are supposed to be constants).

**Enum Miscellaneous Details**

Enums extend the java.lang.Enum class implicitly, so your enum types cannot extend another class.

when an enum contain fields and methods, the definition of fields and methods must always come *after* the list of constants in the enum. Additionally, the list of constants must be terminated by a semicolon;