Tutorial of Enum

> Based on the tutorial of "2020S-Java-A" designed by teaching group in SUSTech
>
> Modified (only change to markdown file) by ZHU Yueming in 2021. April. 19th
>
> Add an enum exercise by ZHU Yueming in 2023. Nov. 10th.

Objective

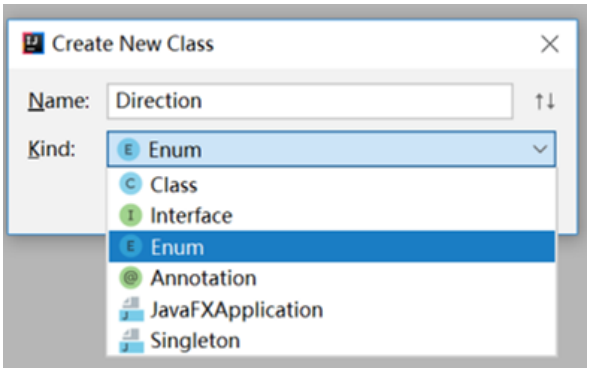
- Learn to use `enum` types
- Learn how to use ArrayList

Part 1: Enumerations

Basic Enum Type

An `enum` type is a special data type that enables 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. For example, a week has seven days (MONDAY to SUNDAY).

A `enum` type is declared using the `enum` keyword, not class. Let's create a new `enum` type Direction with four constants named "NORTH", "SOUTH", "EAST", and "WEST", respectively. In IDEA, creating a new `enum` type is similar to creating a new class. The only difference is to select `Enum` in the dropdown list.




```
public enum Direction {
    NORTH, SOUTH, EAST, WEST // semicolon unnecessary
}
```

Variables of this `enum` type Direction can only receive the values of the four `enum` constants. For example, the following code creates an object of this `enum` type.

```
public class DirectionTest {
    public static void main(String[] args) {
        Direction d = Direction.EAST;
        System.out.println(d);
    }
}
```

The above code prints `EAST`. The last statement in the main method is equivalent to `System.out.println(d.toString())`. The `toString()` method returns the name of the `enum` constant EAST.

In the code, we cannot create an object of the `enum` type using the "new" operator with a constructor call. If you compile the following code, you will receive the error message "Enum types cannot be instantiated".

```
public Direction d = new Direction();
```

Composition in Enum Type

An `enum` type variable can be passed as an argument to a `switch` statement.

```
public class DirectionTest {

  private Direction d;

  public DirectionTest(Direction d) {
    this.d = d;
  }

  public Direction getDirection() {
    return d;
  }

  public static void main(String[] args) {
    DirectionTest test = new DirectionTest(Direction.EAST);
    switch(test.getDirection()) {
      case EAST: // must be unqualified name of the enum constant
        System.out.println("Countries in the east: Japan, Korea");
        break;
      case WEST:
        System.out.println("Countries in the west: US, Germany");
        break;
      case SOUTH:
        System.out.println("Countries in the south: Australia, New Zealand");
        break;
      case NORTH:
        System.out.println("Countries in the north: Russia, Mongolia");
        break;
```

```
    }
  }
}
```

Add attributes in Enum Class

Continue to the desin above, we can add attributes, constructors and other object methods in Enum class.

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST;

    //Add attributes
    private int row;
    private int col;

    //Add constructor:
    Direction(int row, int col) {
        this.row = row;
        this.col = col;
    }

    //Add methods
    public int getRow() {
        return row;
    }

    public void setRow(int row) {
        this.row = row;
    }

    public int getCol() {
        return col;
    }

    public void setCol(int col) {
        this.col = col;
    }

  @Override
    public String toString() {
        return String.format("%s (%d, %d)",this.name(),this.row, this.col);
    }
}
```

The method above has a problem that there is only one constructor with two paramenters but there is no arguments passed in either constants. To solve this problem, we need passing arguments as below:

```
NORTH(-1,0), SOUTH(1,0), EAST(0,1), WEST(0,-1);
```

Values method and EnumSet

The values() method is a static method that is automatically generated by the compiler to return an array of the `enum` constants (an array of references to the objects of the `enum` type).

```
for (Direction d:Direction.values()) {
            System.out.println(d);
        }
```

The generic class `EnumSet` is static method range() returns a collection of the `enum` constants in the range specified by two endpoints. In the above code, range() takes two `enum` constants as arguments. The first constant should be declared before the second (the ordinal() method of a `enum` constant can return the position of the constant in all declared constants). If this constraint is violated (for example, when `EnumSet.range(Direction.EAST,Direction.NORTH)` is used in the code), an `java.lang.IllegalArgumentException` will be thrown.

```
for (Direction d: EnumSet.range(Direction.NORTH,Direction.EAST)) {
            System.out.println(d);
       }
```

Exercise 1： Enum and Composition

Design a enum class named **BookStatus**, and it has three constants:

```
IDLE, BORROWED, OVERDUE;
```

Design a class named **Book**, and it has three private fields.

- int id; //The id of Book
- String name;// The name of book
- BookStatus status;// The status of book

Add one static class member:

- static int count; //It records how many Books have been create.

Add one **constructor** only with the **name** of book, in constructor set its id as the created order of book, and give status a default value **IDLE**.

Add all **getter** and **setter** method of those fields.

Add a **toString** to return a book instance with the format below:

```
[id]: [name] [status]
```

Such as:

`1: Java IDLE`

Design a class named **BookTest**, and complete the methods to finish the requirements below:

- In **borrowBook** method:

```
public static void borrowBook(Scanner in, Book[] books)
```

  - List the books which can be borrowed. (IDLE)
  - Input an book id, which means the id of book would be borrowed.
  - If the status of book is IDLE, then change it status to borrowed and print promp message.
- In **returnBook** method:

```
 public static void returnBook(Scanner in, Book[] books)
```

  - List the books which can be returned. (BORROWED and OVERDUE)
  - Input an book id, which means the id of book would be returned.
  - If the status of book is not IDLE, then change it status to IDLE and print promo message.
- In **overdueAll** method:

```
public static void overdueAll(Book[] books)
```

  - Change the books which status are BORROWED to OVERDUE, and print them.

Code to be complete:

```
public class BookTest {
    public static void main(String[] args) {
        Book[] books = new Book[5];
        books[0] = new Book("Java");
        books[1] = new Book("C++");
        books[2] = new Book("Python");
        books[3] = new Book("Javascript");
        books[4] = new Book("C#");

        Scanner in = new Scanner(System.in);
        while(true){
            System.out.println("Please select operation: " +
                    "1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process");
            int op = in.nextInt();
            if (op == 0)
                break;
            switch (op) {
                case 1:
                    borrowBook(in, books);
                    break;
```

```
                case 2:
                    returnBook(in, books);
                    break;
                case 3:
                    overdueAll(books);
                    break;
            }
        }
    }

    public static void borrowBook(Scanner in, Book[] books) {
       //todo: borrow book
    }

    public static void returnBook(Scanner in, Book[] books) {
        //todo: return book
    }

    public static void overdueAll(Book[] books) {
        //todo: overdueAll
    }
}
```

Sample output:

```
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
1
1: Java IDLE
2: C++ IDLE
3: Python IDLE
4: Javascript IDLE
5: C# IDLE
Please input the book id:
2
borrow C++ successfully
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
1
1: Java IDLE
3: Python IDLE
4: Javascript IDLE
5: C# IDLE
Please input the book id:
4
borrow Javascript successfully
```

```
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
2
2: C++ BORROWED
4: Javascript BORROWED
Please input the book id:
2
return 2: C++ IDLE successfully
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
3
Setting all borrowed book to overdue
4: Javascript OVERDUE
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
1
1: Java IDLE
2: C++ IDLE
3: Python IDLE
5: C# IDLE
Please input the book id:
5
borrow C# successfully
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
2
4: Javascript OVERDUE
5: C# BORROWED
Please input the book id:
4
return 4: Javascript IDLE successfully
Please select operation: 1. Borrow. 2. Return. 3. Overdue. 0. To Stop the
process
0
```

Part 2: ArrayList

How to use ArrayList

An ArrayList servers as a resizable-array that stores data based on an array structure. Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.

The elements in an ArrayList must be objects of a class, and they cannot be primitive data types. However, we can use wrapper classes for primitive data types to store data.

Create an Integer List

```
ArrayList<Integer> numbers = new ArrayList<>();
```

You should write `Integer` in `<>` instead of `<int>`, becaue `Integer` is the wrapper class of `int`.

isEmpty() and size()

```
System.out.println(numbers.isEmpty());
System.out.println(numbers.size());
```

add value into ArrayList

```
numbers.add(1);
numbers.add(3);
numbers.add(5);
numbers.add(7);
```

Traverse ArrayList

```
for (int i = 0; i < numbers.size(); i++) {
     System.out.println(numbers.get(i));
}

for (int e:numbers) {
     System.out.println(e);
}
```

Exercise 1: Manage multiple circle objects by ArrayList.

Continue to the exercise above, we can use an array or an `ArrayList` to store circles.

In the main method, create an arrayList with a Circle type, to store many objects of Circle. Add the following code in main method.

```
ArrayList<Circle> circleList=new ArrayList<>();
Circle c1 = new Circle(1,2,3);
circleList.add(c1);
System.out.println(circleList.get(0));
```

Sample output:

```
Circle #1: radius = 1.00, x = 2.00, y = 3.00
```

Add the following code at the end of main method.

```
for (int i = 1; i < 5; i++) {
      Circle c = new Circle(i, Math.random() * 5, Math.random() * 5);
      circleList.add(c);
}
System.out.println("---Begin to print the circle list---");
for (Circle c: circleList) {
     System.out.println(c);
}
```

Sample output:

```
Circle #1: radius = 1.00, x = 2.00, y = 3.00
---Begin to print the circle list---
Circle #1: radius = 1.00, x = 2.00, y = 3.00
Circle #2: radius = 1.00, x = 0.54, y = 2.66
Circle #3: radius = 2.00, x = 1.29, y = 2.42
Circle #4: radius = 3.00, x = 4.46, y = 3.92
Circle #5: radius = 4.00, x = 4.00, y = 1.93
```