Object Oriented Programming in Java

enum Classes

Suhel Hammoud

enum

What are enums?

- enums are classes where all instances are known to the compiler.
- Used for creating types with a fixed set of possible values.
- Created using the `enum` keyword instead of `class`.
- enum constants are listed in the body, separated by commas.
- No instances can be created outside of enum constants.

```
public enum DayOfWeek {
  MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
```

- All enums implicitly extend `java.lang.Enum` and cannot have subclasses.

Accessing, Evaluating, and Comparing enums

- enum values can be used as constants.
- Use == to compare enum instances.

```
DayOfWeek weekStart = DayOfWeek.MONDAY;

if (weekStart == DayOfWeek.MONDAY) {
  System.out.println("The week starts on Monday.");
}
```

Accessing, Evaluating, and Comparing enums

- Use `switch` for actions based on enum values.
- Switch expressions ensure **exhaustiveness** (all enum values are handled).

```
DayOfWeek someDay = DayOfWeek.FRIDAY;

switch (someDay) {
  case MONDAY -> System.out.println("start of week");
  case TUESDAY, WEDNESDAY, THURSDAY -> System.out.println("middle");
  case FRIDAY -> System.out.println("weekend is near.");
  case SATURDAY, SUNDAY -> System.out.println("Weekend");
  default -> throw new AssertionError("Should not happen");
}
```

Adding memebers to enum

- enums can have constructors, methods, and fields.
- Add a ; after the enum constants list to define members.

```
public enum DayOfWeek {
  MONDAY("MON"), TUESDAY("TUE"), WEDNESDAY("WED"),
  THURSDAY("THU"), FRIDAY("FRI"), SATURDAY("SAT"), SUNDAY("SUN");

  private final String abbreviation;

  DayOfWeek(String abbreviation) { this.abbreviation = abbreviation;}

  public String getAbbreviation() { return abbreviation; }
}
```

Using enums as Singletons:

▓▓ Define a single enum constant

```
public enum SomeSingleton {
    INSTANCE;
    // Fields, methods, etc.
}
```

enum Methods

▒▒ Special Methods in enums

- **Instance Methods**:
  - `name()`: Returns the name of the enum constant.
  - `ordinal()`: Returns the position of the enum constant in the declaration.

```
System.out.println(DayOfWeek.MONDAY.name());    // "MONDAY"
System.out.println(DayOfWeek.MONDAY.ordinal()); // "0"
```

enum Methods

Special Methods in enums

- **Static Methods**:
  - `values()`: Returns an array of all enum instances.
  - `valueOf(String)`: Returns an enum instance by name.

```
DayOfWeek[] days = DayOfWeek.values(); // All days
DayOfWeek monday = DayOfWeek.valueOf("MONDAY");
```

- enums implement `Comparable` and are ordered by their ordinal number.

- **Abstract Methods** : each enum constant must provide an implementation.

```
enum MyEnum {
  A() {
    @Override
    void doSomething() {
      System.out.println("a");
    }
  },
  B() {
    @Override
    void doSomething() {
      System.out.println("b");
    }
  };

  abstract void doSomething();
}
```

enum: Precautions

Precautions with enums

- **Changes to enum Constants**:
  - Adding, removing, or renaming enum constants can break code.
  - Review all code using the enum when making changes.
- **Large Number of Instances**:
  - For many instances, consider using a configuration file instead of listing all in code.