

Kotitehtävä 4 Deadline 5.10. klo 11:45

Maksimipisteet 10

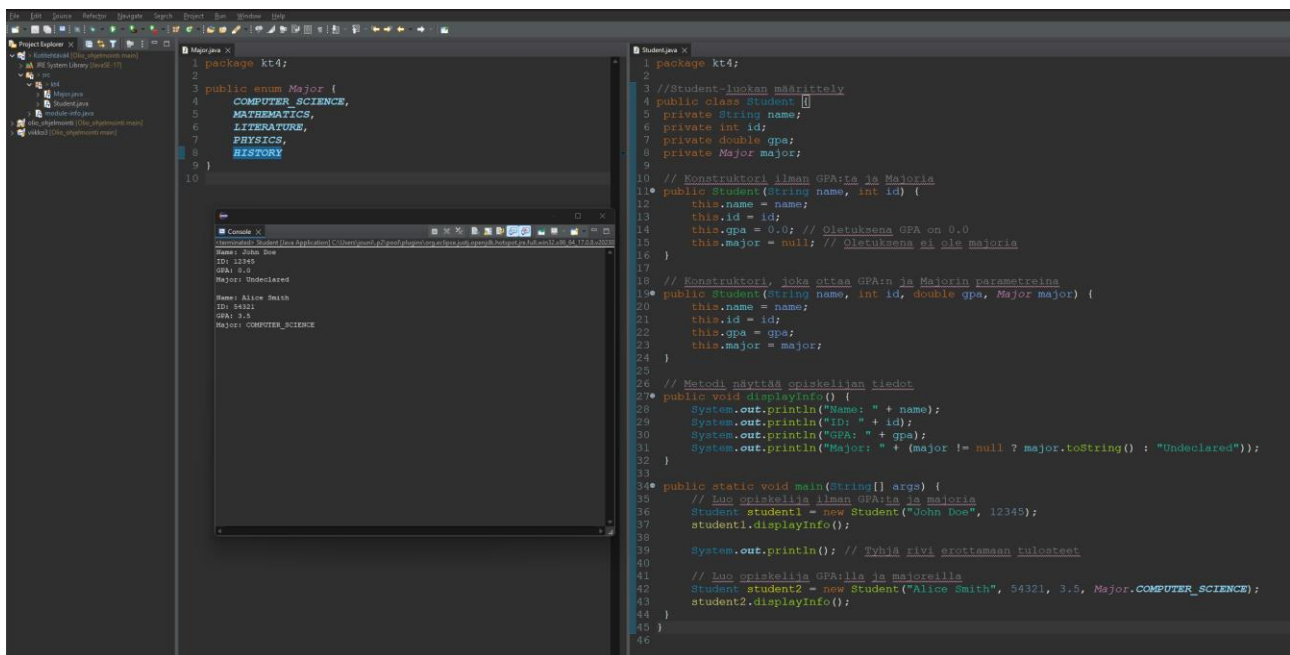
Valitse tehtäviä niin, että saat niistä maksimissaan 10 pistettä

Tehtävä 1: Student luokka, Major enum (2 p)

School Days (Try It Out p. 244)

Create a Student class with a name, an ID number, a grade point average (GPA), and a major area of study. The student's name is a String. The student's ID number is an int value. The GPA is a double value between 0.0 and 4.0. The Major is an enum type, with values such as COMPUTER_SCIENCE, MATHEMATICS, LITERATURE, PHYSICS, and HISTORY.

Every student has a name and an ID number, but a brand-new student might not have a GPA or a major. Create constructors with and without GPA and Major parameters. As usual, create a separate class that makes use of your new Student class.



```
1 package kt4;
2
3 public enum Major {
4     COMPUTER_SCIENCE,
5     MATHEMATICS,
6     LITERATURE,
7     PHYSICS,
8     HISTORY
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

```
1 package kt4;
2
3 //Student-luokan määrittely
4 public class Student {
5     private String name;
6     private int id;
7     private double gpa;
8     private Major major;
9
10    // Konstruktori ilman GPA:ta ja Majoria
11    public Student(String name, int id) {
12        this.name = name;
13        this.id = id;
14        this.gpa = 0.0; // Oletuksena GPA on 0.0
15        this.major = null; // Oletuksena ei ole majoria
16    }
17
18    // Konstruktori, joka ottaa GPA:n ja Majorin parametreina
19    public Student(String name, int id, double gpa, Major major) {
20        this.name = name;
21        this.id = id;
22        this.gpa = gpa;
23        this.major = major;
24    }
25
26    // Metodi näyttää opiskelijan tiedot
27    public void displayInfo() {
28        System.out.println("Name: " + name);
29        System.out.println("ID: " + id);
30        System.out.println("GPA: " + gpa);
31        System.out.println("Major: " + (major != null ? major.toString() : "Undeclared"));
32    }
33
34    public static void main(String[] args) {
35        // Luo opiskelija ilman GPA:ta ja majoria
36        Student student1 = new Student("John Doe", 12345);
37        student1.displayInfo();
38
39        System.out.println(); // Tyhjä rivi erottamaan tulosteet
40
41        // Luo opiskelija GPA:lla ja majorilla
42        Student student2 = new Student("Alice Smith", 54321, 3.5, Major.COMPUTER_SCIENCE);
43        student2.displayInfo();
44    }
45 }
46
```

```
1 Name: John Doe
2 ID: 12345
3 GPA: 0.0
4 Major: Undeclared
5
6 Name: Alice Smith
7 ID: 54321
8 GPA: 3.5
9 Major: COMPUTER_SCIENCE
```

Tehtävä 2: AirplaneFlight luokka, Airport enum (3p)

Tässä tehtävässä tehdään osa kirjan Flight of Fancy tehtävästä. Tässä harjoituksessa käytetään pelkästään suomalaisia lentokenttiä, joten sinun ei tarvitse välittää aikaeroista. Jatketaan tämä tehtävä loppuun myöhemmin. Saat tietysti kokeilla tehdä tehtävää niin kuin se on esitetty kirjassa.

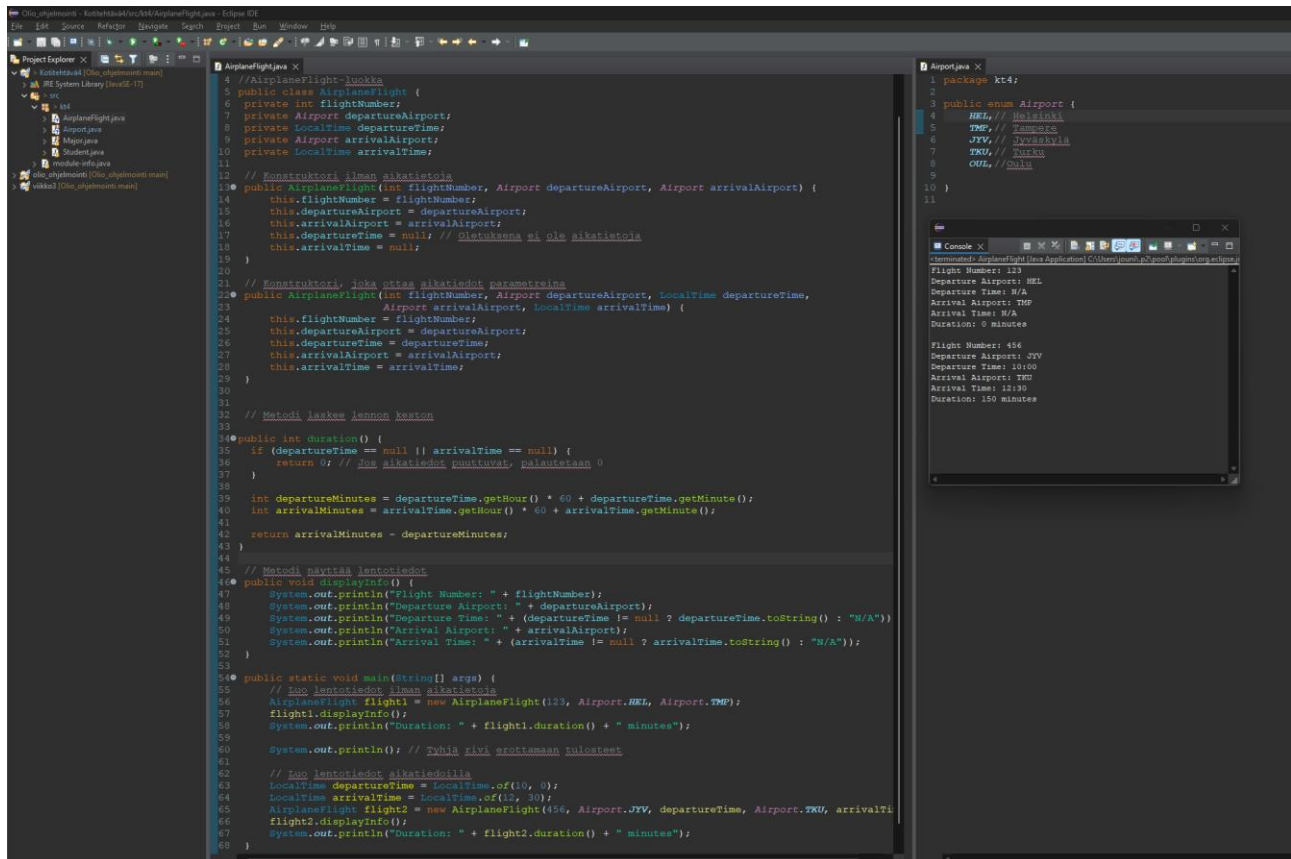
Flight of Fancy (Try It Out p. 244)

Create an AirplaneFlight class with a flight number, a departure airport, the time of departure, an arrival airport, and a time of arrival. The flight number is an int value. The departure and arrival airport fields belong to an Airport enum type, with values corresponding to some of the official IATA airport codes. Create 6 airports, for example, HEL, TMP, JYV, TKU and OUL.

Every flight has a number, a departure airport, and an arrival airport. But some flights might not have departure and arrival times. All the departure and arrival times are given in Finnish time. Create

constructors with and without departure and arrival time parameters. Create a separate class that makes use of your new AirplaneFlight class.

Create a subclass that has a method named duration. The duration method, which has no parameters, returns the amount of time between the flight's departure time and arrival time.



```
4 //AirplaneFlight-luokka
5 public class AirplaneFlight {
6     private int flightNumber;
7     private Airport departureAirport;
8     private LocalDateTime departureTime;
9     private Airport arrivalAirport;
10    private LocalDateTime arrivalTime;
11
12    // Konstruktori ilman aikataistoa
13    public AirplaneFlight(int flightNumber, Airport departureAirport, Airport arrivalAirport) {
14        this.flightNumber = flightNumber;
15        this.departureAirport = departureAirport;
16        this.arrivalAirport = arrivalAirport;
17        this.departureTime = null; // Etukäteen ei ole aikataistoa
18        this.arrivalTime = null;
19    }
20
21    // Konstruktori, joka ottaa aikataistot parametreina
22    public AirplaneFlight(int flightNumber, Airport departureAirport, LocalDateTime departureTime,
23        Airport arrivalAirport, LocalDateTime arrivalTime) {
24        this.flightNumber = flightNumber;
25        this.departureAirport = departureAirport;
26        this.departureTime = departureTime;
27        this.arrivalAirport = arrivalAirport;
28        this.arrivalTime = arrivalTime;
29    }
30
31    // Metodi laskee lennon keston
32
33    public int duration() {
34        if (departureTime == null || arrivalTime == null) {
35            return 0; // Jos aikataistot puuttuvat, palautetaan 0
36        }
37
38        int departureMinutes = departureTime.getHour() * 60 + departureTime.getMinute();
39        int arrivalMinutes = arrivalTime.getHour() * 60 + arrivalTime.getMinute();
40
41        return arrivalMinutes - departureMinutes;
42    }
43
44    // Metodi näyttää lentotiedot
45    public void displayInfo() {
46        System.out.println("Flight Number: " + flightNumber);
47        System.out.println("Departure Airport: " + departureAirport);
48        System.out.println("Departure Time: " + (departureTime != null ? departureTime.toString() : "N/A"));
49        System.out.println("Arrival Airport: " + arrivalAirport);
50        System.out.println("Arrival Time: " + (arrivalTime != null ? arrivalTime.toString() : "N/A"));
51    }
52
53    public static void main(String[] args) {
54        // Luo lentotiedot ilman aikataistoa
55        AirplaneFlight flight1 = new AirplaneFlight(123, Airport.HEI, Airport.TMP);
56        flight1.displayInfo();
57        System.out.println("Duration: " + flight1.duration() + " minutes");
58
59        System.out.println(); // Tyhjä rivi erotamaan tulokset
60
61        // Luo lentotiedot aikataistolla
62        LocalDateTime departureTime = LocalDateTime.of(10, 30);
63        LocalDateTime arrivalTime = LocalDateTime.of(12, 30);
64        AirplaneFlight flight2 = new AirplaneFlight(456, Airport.JYV, departureTime, Airport.TKU, arrivalTime);
65        flight2.displayInfo();
66        System.out.println("Duration: " + flight2.duration() + " minutes");
67    }
68 }
```

```
1 package kt4;
2
3 public enum Airport {
4     HEI, // Helsinki
5     TMP, // Tampere
6     JYV, // Jyväskylä
7     TKU, // Turku
8     CGU, // Chgo
9 }
10
11
```

```
Flight Number: 123
Departure Airport: HEI
Departure Time: N/A
Arrival Airport: TMP
Arrival Time: N/A
Duration: 0 minutes

Flight Number: 456
Departure Airport: JYV
Departure Time: 10:30
Arrival Airport: TKU
Arrival Time: 12:30
Duration: 120 minutes
```

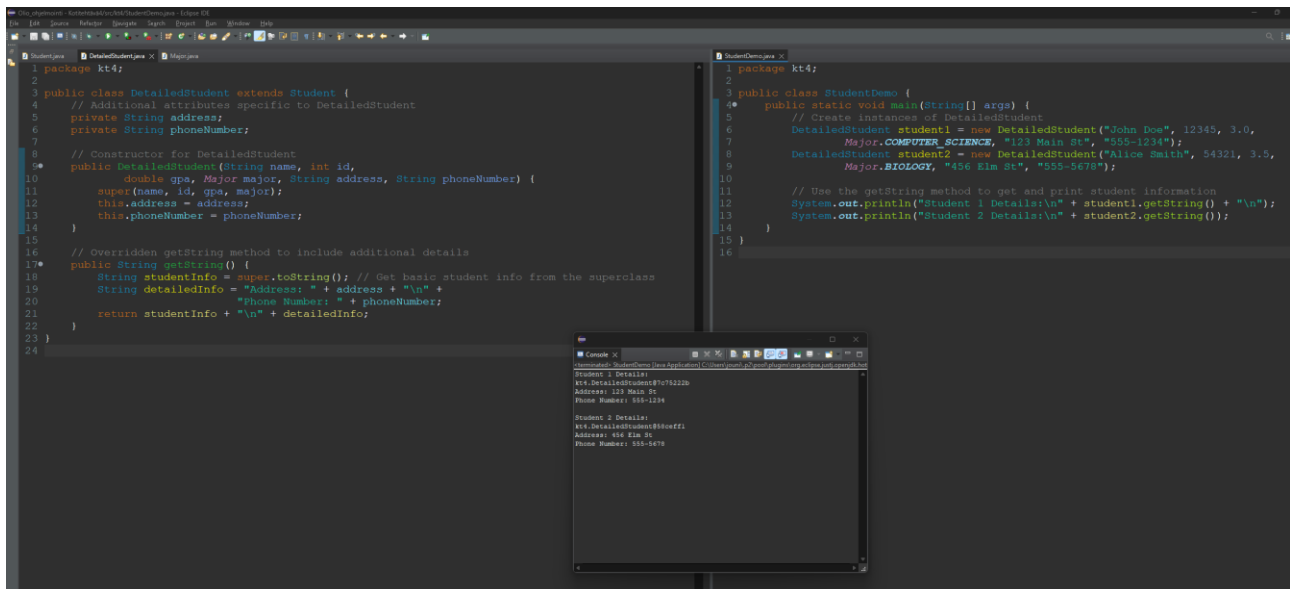
Tehtävä 3: getString method inside a subclass (2p)

Student Showcase (Try It Out p. 252)

In the first exercise, you created your own Student class. Create a subclass that has a method named getString. Like the display method in this chapter's TemperatureNice class, the getString method creates a nice-looking String representation of its object. But unlike the TemperatureNice class's display method, the getString method doesn't print that String representation on the screen. Instead, the getString method simply returns that String representation as its result.

(In a way, a getString method is much more versatile than a display method. With a display method, all you can do is show a String representation on the screen. But with a getString method, you can create a String representation and then do whatever you want with it.)

Create a separate class that creates some instances of your new subclass and puts their getString methods to good use.



```
1 package kt4;
2
3 public class DetailedStudent extends Student {
4     // Additional attributes specific to DetailedStudent
5     private String address;
6     private String phoneNumber;
7
8     // Constructor for DetailedStudent
9     public DetailedStudent(String name, int id,
10         double gpa, Major major, String address, String phoneNumber) {
11         super(name, id, gpa, major);
12         this.address = address;
13         this.phoneNumber = phoneNumber;
14     }
15
16     // Overridden toString method to include additional details
17     public String toString() {
18         String studentInfo = super.toString(); // Get basic student info from the superclass
19         String detailedInfo = "Address: " + address + "\n" +
20             "Phone Number: " + phoneNumber;
21         return studentInfo + "\n" + detailedInfo;
22     }
23 }
24
```

```
1 package kt4;
2
3 public class StudentDemo {
4     public static void main(String[] args) {
5         // Create instances of DetailedStudent
6         DetailedStudent student1 = new DetailedStudent("John Doe", 12345, 3.0,
7             Major.COMPUTER_SCIENCE, "123 Main St", "555-1234");
8         DetailedStudent student2 = new DetailedStudent("Alice Smith", 54321, 3.5,
9             Major.BIOLOGY, "456 Elm St", "555-5678");
10
11         // Use the toString method to get and print student information
12         System.out.println("Student 1 Details:\n" + student1.toString() + "\n");
13         System.out.println("Student 2 Details:\n" + student2.toString());
14     }
15 }
16
```

```
Student 1 Details:
kt4.DetailedStudent@9c7b22b
Address: 123 Main St
Phone Number: 555-1234

Student 2 Details:
kt4.DetailedStudent@8b0c0ff1
Address: 456 Elm St
Phone Number: 555-5678
```

Tehtävä 4: Painoindeksi (2p)

Tee luokka Henkilo. Luokalla on seuraavat muuttujat: nimi, paino, pituus. Luo Henkilo niminen konstruktori, jolla on parametrina nimi, alusta muut muuttujat arvolla 0.

Luo luokalle setPituus, setPaino ja painoindeksi metodit. Painoindeksi lasketaan $\text{paino}/\text{pituus}^2$. Pituus ilmoitetaan metreinä ja paino kiloina.

Luo luokalle vielä display metodi, jossa tulostat kunkin henkilön tiedot seuraavasti:

Nimi, painoindeksisi on painoindeksi

Eli esim.

Matti, painoindeksisi on 26,54.

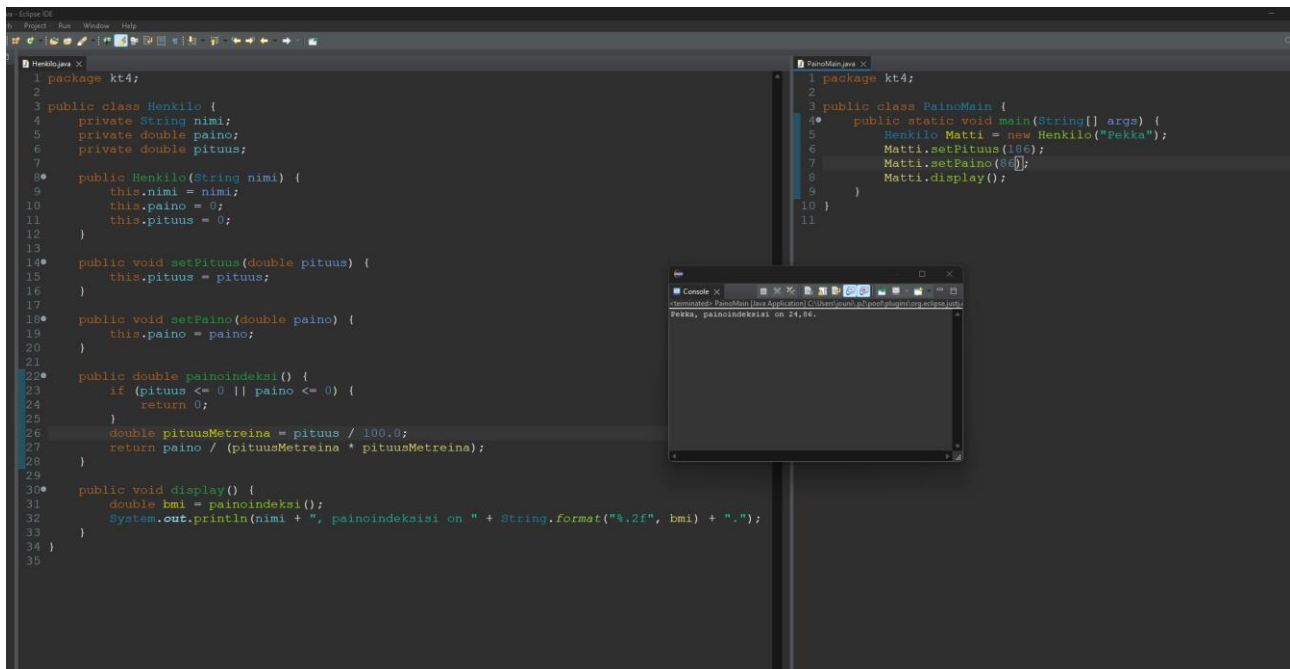
Teen sen jälkeen main-funktio, jossa käytät äsken luotuja metodeja, esim.

Henkilo matti = new Henkilo("Matti");

matti.setPituus(180);

matti.setPaino(86);

matti(display);



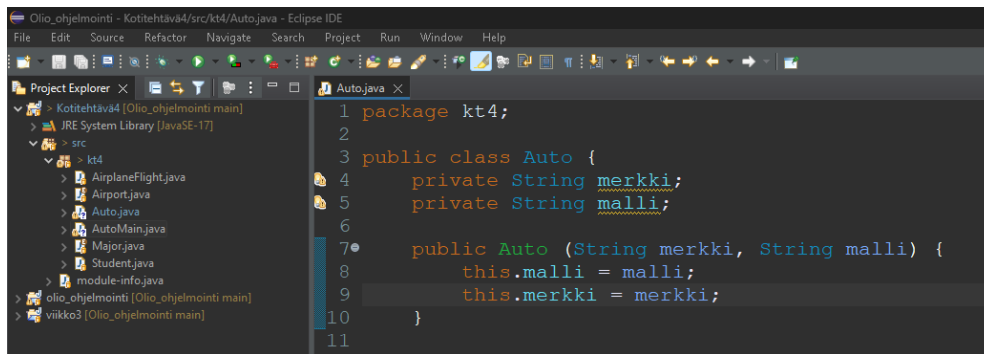
Tehtävä 5: Selitä (3p selityksistä ja 3p esimerkeistä, yht. 6p)

Selitä omin sanoin seuraavat termit, mitä ne tarkoittavat ja milloin niitä käytetään. Anna käytöstä esimerkki.

a. Konstruktori

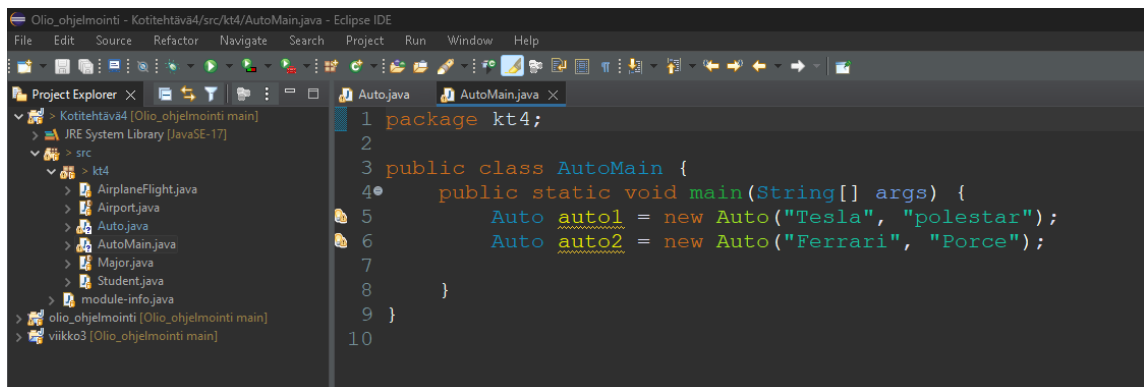
Konstruktori on erityinen metodi, joka suoritetaan, kun luot uuden oliion tietylle luokalle. Se on vastuussa luokan olioiden alustamisesta ja asettaa niiden alkutilan. Konstruktori on nimetty samalla nimellä kuin luokka, ja se voi ottaa parametreja, jotka määrittelevät sen, miten olio alustetaan.

Esimerkissäni **Auto**-luokassa on seuraava konstruktori:



```
1 package kt4;
2
3 public class Auto {
4     private String merkki;
5     private String malli;
6
7     public Auto (String merkki, String malli) {
8         this.malli = malli;
9         this.merkki = merkki;
10    }
11 }
```

Tämä konstruktori ottaa kaksi parametria: **merkki** ja **malli**. Kun luon uuden **Auto**-olion, kuten teen **AutoMain**-metodissa:



```
1 package kt4;
2
3 public class AutoMain {
4     public static void main(String[] args) {
5         Auto auto1 = new Auto("Tesla", "polestar");
6         Auto auto2 = new Auto("Ferrari", "Porce");
7     }
8 }
9
10
```

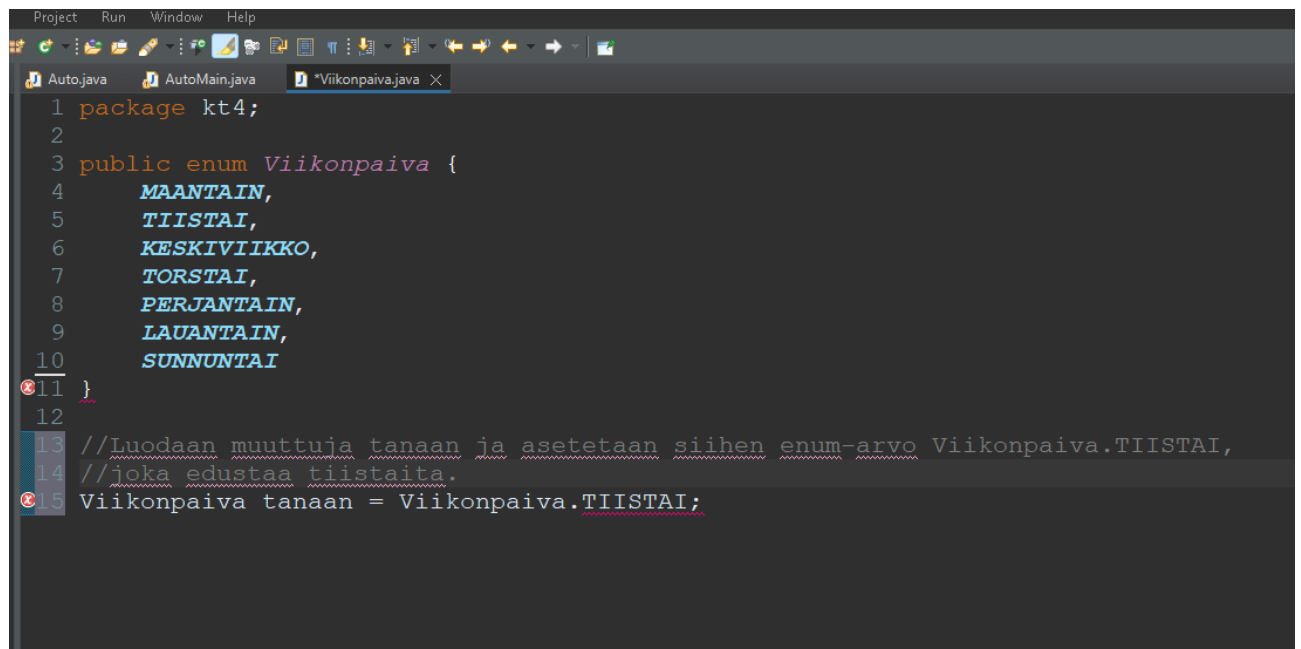
Nämä parametrit (esimerkiksi "Toyota" ja "Corolla") lähetetään konstruktorille. Konstruktori käyttää sitten näitä parametreja asettaakseen **merkki**- ja **malli**-kentät oikeille arvoille. Tämä tarkoittaa, että jokaiselle luodulle **Auto**-oliolle annetaan omat merkki ja malli, jotka määrittävät luonnin yhteydessä.

Konstruktori mahdollistaa siis olioiden alkutilan määrittämisen ja varmistaa, että niiden tietoja ei unohdeta tai jää vahingossa asettamatta.

b. Enum

Enum (lyhenne sanoista "enumerated type") on erityinen tyyppi Java-ohjelmointikielessä, joka koostuu määrätystä joukosta vakioita. Enumit luodaan luokan tavoin, mutta ne voivat sisältää vain ennalta määritettyjä vakioita, joita kutsutaan usein "enumerointiarvoiksi" tai "enum-arvoiksi". Enumit ovat hyödyllisiä, kun halutaan rajoittaa tiettyjen arvojen käyttöä ja tehdä koodista selkeämpää ja ymmärrettävämpää.

Alla on esimerkki enumin käytöstä:



```
1 package kt4;
2
3 public enum Viikonpaiva {
4     MAANTAIN,
5     TIISTAI,
6     KESKIVIIKKO,
7     TORSTAI,
8     PERJANTAIN,
9     LAUANTAIN,
10    SUNNUNTAI
11 }
12
13 //Luodaan muuttuja tanaan ja asetetaan siihen enum-arvo Viikonpaiva.TIISTAI,
14 //joka edustaa tiistaita.
15 Viikonpaiva tanaan = Viikonpaiva.TIISTAI;
```

Tässä **tanaan**-muuttujalle annetaan arvoksi **Viikonpaiva.TIISTAI**, joka vastaa tiistain enum-arvoa. Enumit tekevät koodista helposti luettavaa ja ymmärrettävää, koska ne rajoittavat sallittujen arvojen joukkoa ja tarjoavat selkeät nimet näille arvoille. Ne ovat erityisen hyödyllisiä tilanteissa, joissa on tiettyjä kiinteitä arvoja, jotka eivät saisi muuttua suorituksen aikana, kuten viikonpäivät, kuukaudet tai lippuarvot.

c. Super

Java-ohjelmointikielessä "super" on avainsana, joka viittaa yliliuokan (superclass) ominaisuuksiin ja metodeihin. Se käytetään yleensä aliluokassa (subclass) ilmaisemaan, että halutaan kutsua yliliuokan konstruktoria tai metodia.

Alla on esimerkki siitä, miten "super" toimii

