



Java Lab – Static keyword in Java

IMPORTANT! Save all your work to a safe location such as oneDrive.

Create a folder for SDPD into which you will save all your work for this module, arranged how you wish. Ideally you should create a folder each week for your lab exercises. Note that you should create **a separate file** for each exercise.

Exercise 1

Write a class called *Exchange* that has a static method called *euroToDollar* that converts Euros to Dollars (assume that one euro buys you \$1.1914447 dollars). Test this class out by calling the method from a separate program file with a main method. Your Exchange class can be similar to as shown below:

```
1 public class Exchange
2 {
3     public static double euroToDollar(double euro)
4     {
5         return 1.1914447 * euro;
6     }
7
8 }
```

Your class with your main method can be similar to as shown here:

```
1 public class Exercisel
2 {
3     public static void main(String[] args)
4     {
5         System.out.println(Exchange.euroToDollar(100));
6     }
7 }
```

Note that an instance of the class is not required – the method can be invoked (called) using the class name.

Exercise 2

Create a class file called *Currency* that has multiple static methods to convert the following:

- Euros To Dollars
- Dollars to Euros
- Euros to Sterling
- Sterling to Euros
- Sterling to Dollars
- Dollars to Sterling

You can use the following rates for this:

1 Euro buys 1.1914447 Dollars

1 Euro buys 0.8660 Sterling

1 Dollar buys 0.7274 Sterling

Test your program using a separate file with a main method to confirm that all conversions work correctly.

Use *printf* in your output to ensure only 2 decimal places appear in your output.

```
C:\Windows\system32\cmd.exe
100 Euro to Dollar: 119.14
100 Dollar to Euro: 83.93
100 Euro to Sterling: 86.60
100 Sterling to Euro: 115.47
100 Sterling to Dollar: 137.48
100 Dollar to Sterling: 72.74
Press any key to continue . . .
```

Exercise 3

Create a class called Area that has three static methods for calculating the areas of the following geometric shapes:

- circles
- rectangles
- cylinders

Here are the formulas for calculating the area of the shapes.

Area of a circle: $\text{Area} = \pi r^2$

where π is 3.1415 and r is the circle's radius

Area of a rectangle: $\text{Area} = \text{Width} * \text{Length}$

Area of a cylinder: $\text{Area} = \pi r^2 h$

where π is 3.1415, r is the radius of the cylinder's base, and h is the cylinder's height

Demonstrate the class in a complete program, eg:

```
C:\Windows\system32\cmd.exe
The area of a circle with a radius of 20.0 is 1256.64
The area of a rectangle with a length of 10 and a width of 20 is 200.00
The area of a cylinder with a radius of 10.0 and a height of 15.0 is 4712.39
Press any key to continue . . .
```

Exercise 4

Create a class file called *Person* that will allow for the creation of an object that contains a first name and a surname, using a constructor that allows for both values to be passed as arguments. The class should also contain a static field called *objectCount* so that each time an object is created using the class, the field *objectCount* increases by 1. In your main method, create 2 instances of the *Person* object, as shown below:

Person Class File:

```
public class Person
{
    private String firstName;
    private String lastName;
    private static int instanceCount = 0;

    public Person(String f, String l)
    {
        firstName = f;
        lastName = l;
        instanceCount++;
    }

    public int getInstanceCount()
    {
        return instanceCount;
    }
}
```

Note that the constructor also increments the *instanceCounter* static field, each time it is invoked.

Main Method:

```
1 public class Exercise4
2 {
3     public static void main(String[] args)
4     {
5         Person a = new Person("Grace", "Hopper");
6         Person b = new Person("Bill", "Gates");
7
8         System.out.println(a.getInstanceCount());
9     }
10 }
```

Note that even though the object “a” was only invoked once, it still outputs the value of *instanceCount* as 2

Amend your class so that it also has a *static method* that can be called from the main method using the following:

```
System.out.println(Person.count());
```

This method should display the number of objects created.

Create two additional person objects (for a total of 4) and test that your count methods work as expected.

Exercise 5

Create a class file called *Account* for a bank that will allow for the creation of an object that contains a first name, surname and a deposit for each account holder.

Create 4 account objects for the following 4 persons:

Grace Hopper, who deposits 100

Bill Gates, who deposits 300

Ada Lovelace, who deposits 400

James Gosling, who deposits 230

The class should have a constructor that accepts 3 values – the first name, last name, and the deposit amount. The class should also have a static field called *vault*, that stores the total value of all the deposits. The vault value can also be updated by the constructor (as each object is being created).

In addition, your class should have the following static method:

`getVault()` – this should return the value in the vault, eg:

```
C:\Windows\system32\cmd.exe
The vault currently has 1030.00
```

The *Account* class should also have a *withdraw* method (not a static method). The *withdraw* method should work as follows:

It will allow the account holder to withdraw money from their account, as long as it is available in the vault. For example, Grace, who has only deposited 100 euros, can withdraw 500 euros. The method would then output the following:

```
Grace Hopper has requested to withdraw 500.0 from the vault,
Grace Hopper's current balance is 100.0
TRANSACTION COMPLETE
Grace Hopper's balance is now -400.0
There is now 530.0 in the vault.
- - - - -
```

However, if the withdraw amount is greater than what is in the vault, then the withdraw request will be refused, eg:

```
Grace Hopper has requested to withdraw 1500.0 from the vault,
but there is only 1030.0 in the vault.
NO TRANSACTION
```

Test this method in your main method with the following, eg:

```
grace.withdraw(150);  
bill.withdraw(100);  
ada.withdraw(1000);  
james.withdraw(50);  
grace.withdraw(99);
```

Your results should be as follows:

cmd: C:\Windows\system32\cmd.exe

```
The vault currently has 1030.00  
  
Grace Hopper has requested to withdraw 150.0 from the vault,  
Grace Hopper's current balance is 100.0  
TRANSACTION COMPLETE  
Grace Hopper's balance is now -50.0  
There is now 880.0 in the vault.  
- - - - -  
  
Bill Gates has requested to withdraw 100.0 from the vault,  
Bill Gates's current balance is 300.0  
TRANSACTION COMPLETE  
Bill Gates's balance is now 200.0  
There is now 780.0 in the vault.  
- - - - -  
  
Ada Lovelace has requested to withdraw 1000.0 from the vault,  
but there is only 780.0 in the vault.  
NO TRANSACTION  
  
James Gosling has requested to withdraw 50.0 from the vault,  
James Gosling's current balance is 230.0  
TRANSACTION COMPLETE  
James Gosling's balance is now 180.0  
There is now 730.0 in the vault.  
- - - - -  
  
Grace Hopper has requested to withdraw 99.0 from the vault,  
Grace Hopper's current balance is -50.0  
TRANSACTION COMPLETE  
Grace Hopper's balance is now -149.0  
There is now 631.0 in the vault.  
- - - - -  
  
Press any key to continue . . . █
```

Exercise 6

Create a class file called *Employee* for a company that will allow for the creation of an object that contains a first name, surname and an employee ID for each person.

Create 4 account objects for the following 4 persons:

Grace Hopper, ID 123456

Bill Gates, ID 456789

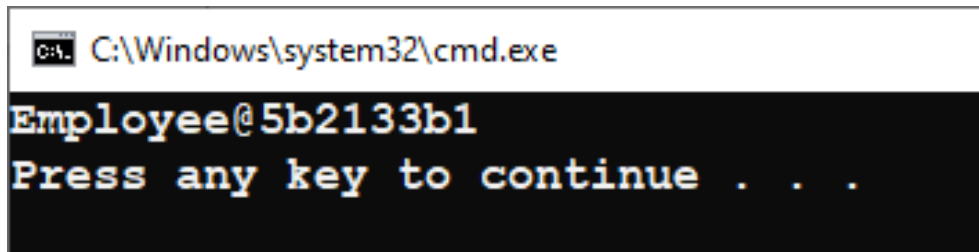
Ada Lovelace, ID 789123

James Gosling, ID 456123

Use the default *toString* method to output to the console for one of the objects, eg:

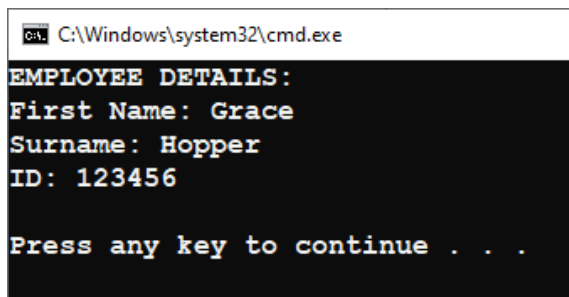
```
System.out.println(grace.toString());
```

It should output something similar to the following:



```
C:\Windows\system32\cmd.exe
Employee@5b2133b1
Press any key to continue . . .
```

In the *Employee* class, write a custom *toString()* method, that will produce the following:



```
C:\Windows\system32\cmd.exe
EMPLOYEE DETAILS:
First Name: Grace
Surname: Hopper
ID: 123456
Press any key to continue . . .
```

Confirm that this method will work on all 4 objects, and also without using *toString()*, eg:

```
System.out.println(grace);
System.out.println(bill);
System.out.println(ada);
System.out.println(james);
```



```
Ca\ Select C:\Windows\system32\cmd.exe

EMPLOYEE DETAILS:
First Name: Grace
Surname: Hopper
ID: 123456

EMPLOYEE DETAILS:
First Name: Bill
Surname: Gates
ID: 456789

EMPLOYEE DETAILS:
First Name: Ada
Surname: Lovelace
ID: 789123

EMPLOYEE DETAILS:
First Name: James
Surname: Gosling
ID: 456123
```

Exercise 7

Create class named Month. The class should have an int field named *monthNumber* that holds the number of the month. For example, January would be 1, February would be 2, and so forth. In addition, provide the following methods:

- A no-arg constructor that sets the monthNumber field to 1.
- A constructor that accepts the number of the month as an argument. It should set the monthNumber field to the value passed as the argument. If a value less than 1 or greater than 12 is passed, the constructor should set monthNumber to 1.
- A constructor that accepts the name of the month, such as “January” or “February” as an argument. It should set the monthNumber field to the correct corresponding value.
- A setMonthNumber method that accepts an int argument, which is assigned to the monthNumber field. If a value less than 1 or greater than 12 is passed, the method should set monthNumber to 1.
- A getMonthNumber method that returns the value in the monthNumber field.
- A getMonthName method that returns the name of the month. For example, if the monthNumber field contains 1, then this method should return “January”.
- A toString method that returns the same value as the getMonthName method.
- A static method that accepts a Month number as an argument, and then outputs a message stating “Then month number you entered was 10” (assuming 10 was entered). If the argument is less than 1 or greater than 12, the message output states “That is not a valid month number”.

Exercise 8

Create a class that will record the votes for candidates for an election. Your class should have 3 private fields for candidate name, political party and number of votes. The class should also have a private static field called *totalVotes* that will record all the votes cast.

Create a program where the main method will allow the user to enter the details for 3 candidates, as shown below.

The program should then allow the user to add additional votes for counting, if required, using a while loop.

The total votes cast should be recorded in the totalVotes static field, and displayed at the end of the program.

The class should also contain a custom *toString()* method that outputs the data about candidate at the end of the program. Finally, the program should display a message stating the winner of the election at the end of the program.

```
C:\Windows\system32\cmd.exe
Enter details for candiate 1:
Enter Name: Hillary
Enter Party: Democrat
Enter Votes: 111

Enter details for candiate 2:
Enter Name: Donald
Enter Party: Repub
Enter Votes: 222

Enter details for candiate 3:
Enter Name: Joe
Enter Party: Democrat
Enter Votes: 333
Enter additional votes? Enter 1, 2, or 3 for candiadte or 0 to end:1
Enter Votes for candidate 1: 234
CANDIDATE DETAILS:
Name: Hillary
Party: Democrat
Votes: 345

Enter additional votes? Enter 1, 2, or 3 for candiadte or 0 to end:3

Enter Votes for candidate 3: 242
CANDIDATE DETAILS:
Name: Joe
Party: Democrat
Votes: 575

Enter additional votes? Enter 1, 2, or 3 for candiadte or 0 to end:0

Total Votes cast: 1142
CANDIDATE DETAILS:
Name: Hillary
Party: Democrat
Votes: 345

CANDIDATE DETAILS:
Name: Donald
Party: Repub
Votes: 222

CANDIDATE DETAILS:
Name: Joe
Party: Democrat
Votes: 575

Joe has been deemed to be elected!
Press any key to continue . . .
```

Exercise 9

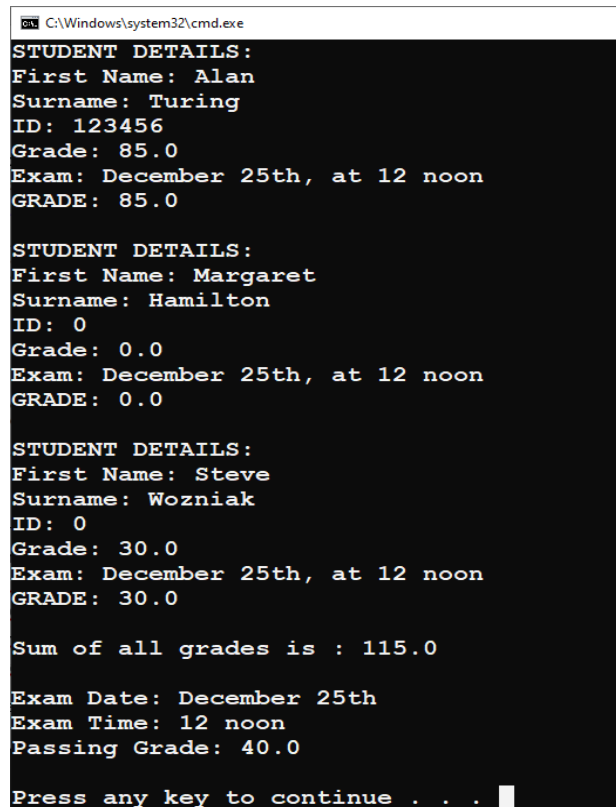
Consider the code shown below:

```
1 public class Exercise9
2 {
3     public static void main(String[] args)
4     {
5         Student alan = new Student("Alan", "Turing", 123456, 85.5);
6         Student margaret = new Student("Margaret", "Hamilton");
7         Student steve = new Student("Steve", "Wozniak", 30.0);
8
9         System.out.println(alan);
10        System.out.println(margaret);
11        System.out.println(steve);
12
13        System.out.print("Sum of all grades is : " + Student.getTotal() + "\n\n");
14        Student.examDetails();
15
16        margaret.setGrade(65);
17        margaret.setId(789456);
18
19        steve.setId(456123);
20    }
21 }
```

Create a program using the code provide above, and write the associated class file to make this work as expected. Note that the Student class that you create should have the following static fields:

- examTime (String) – Default value of: 12noon
- examDate (String) – Default value of: December 25th
- passGrade (double) – Default value of: 40
- allScores (double) – Default value of: 0

After writing the Student class, your output from the main method should produce the following result:



```
C:\Windows\system32\cmd.exe
STUDENT DETAILS:
First Name: Alan
Surname: Turing
ID: 123456
Grade: 85.0
Exam: December 25th, at 12 noon
GRADE: 85.0

STUDENT DETAILS:
First Name: Margaret
Surname: Hamilton
ID: 0
Grade: 0.0
Exam: December 25th, at 12 noon
GRADE: 0.0

STUDENT DETAILS:
First Name: Steve
Surname: Wozniak
ID: 0
Grade: 30.0
Exam: December 25th, at 12 noon
GRADE: 30.0

Sum of all grades is : 115.0

Exam Date: December 25th
Exam Time: 12 noon
Passing Grade: 40.0

Press any key to continue . . .
```

Amend the code so that you can confirm that a grade and ID has been set for Margaret, and that Steve has an ID.

Following that, amend your *setGrade* method so that if a new grade is entered overwriting a previously entered grade, the *allScores* field will still hold the correct value (grades for each 3 students added together). Test this by changing Alan's grade from 85 to 70 using the *setGrade* method.

Finally, create a report method that is used as following in your main method:

```
alan.report();  
margaret.report();  
steve.report();
```

It should produce the following output:

```
Alan Turing has passed the exam!  
Margaret Hamilton has passed the exam!  
Steve Wozniak didn't pass the exam!
```

Exercise 10

Create a programme that will contain a class called “Vehicles” used to record information about different vehicle types. The class should have the following fields:

```
//Instance fields
private String make;
private char type;

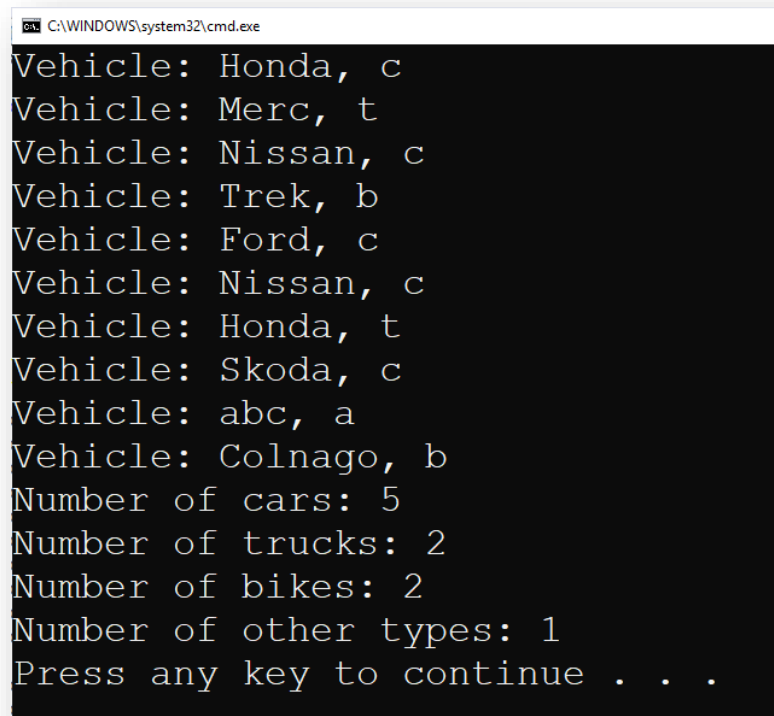
//Static fields
private static int cars;
private static int trucks;
private static int bikes;
private static int other;
```

The class should also have a two-argument constructor used to create the following vehicles:

Vehicle a - Honda – Car
Vehicle b - Merc – Truck
Vehicle c - Nissan – Car
Vehicle d - Trek – Bike
Vehicle e - Ford – Car
Vehicle f - Nissan – Car
Vehicle g - Honda – Truck
Vehicle h - Skoda – Car
Vehicle i - abc – other
Vehicle j - Colnago – Bike

The first argument in the constructor (a String) can be the vehicle make, and the second can be a character (char) with ‘c’ denoting car, ‘t’ denoting truck, ‘b’ denoting bike and all other characters denoting other types. Your program should create these vehicles and also update the count for each type as each object is created – for example, when object **a** is created (“Honda”, ‘c’), this will increment the cars field by 1. Your program should also include a **toString()** method that will output the object string value and object character value.

Your output from your program should look similar to as show:



```
C:\WINDOWS\system32\cmd.exe
Vehicle: Honda, c
Vehicle: Merc, t
Vehicle: Nissan, c
Vehicle: Trek, b
Vehicle: Ford, c
Vehicle: Nissan, c
Vehicle: Honda, t
Vehicle: Skoda, c
Vehicle: abc, a
Vehicle: Colnago, b
Number of cars: 5
Number of trucks: 2
Number of bikes: 2
Number of other types: 1
Press any key to continue . . .
```