Summary JAVA

Sessions No 6(30-10-2022)

- The values that are declared within a function when the function is called are known as an **argument**, Now these arguments are like inputs that we are providing to our functions, Now Function can receive these arguments by defining *their parameters*, and further, it can use them.
  Normally Functions *return* some data at the end, So while defining a function we need to tell the datatype that it will return. For the below example, it is “**int**”.
  Here we are calling this “*myfun*” from the “*main()*” so here it depends on the main() if they wanted to store the *return value* and use it further. In this case, we have stored the return value in variable ***i.***
  The output of the below program will be **25**.

```
public class function {
  public static void main(String[] args) {
        int i =myfun(5);
        System.out.println(i);
  }

  public static int myfun(int x){
    return x*x;
  }
}
```

- In JAVA we have **float** and **double** data types to store the decimal values, They are made to store the values in decimals. But the difference between them is **double** reserves 8 bytes of space in RAM, and float reserves only 4 bytes of space. To assign a value in the float variable, we will have to add “**f**” in the last.

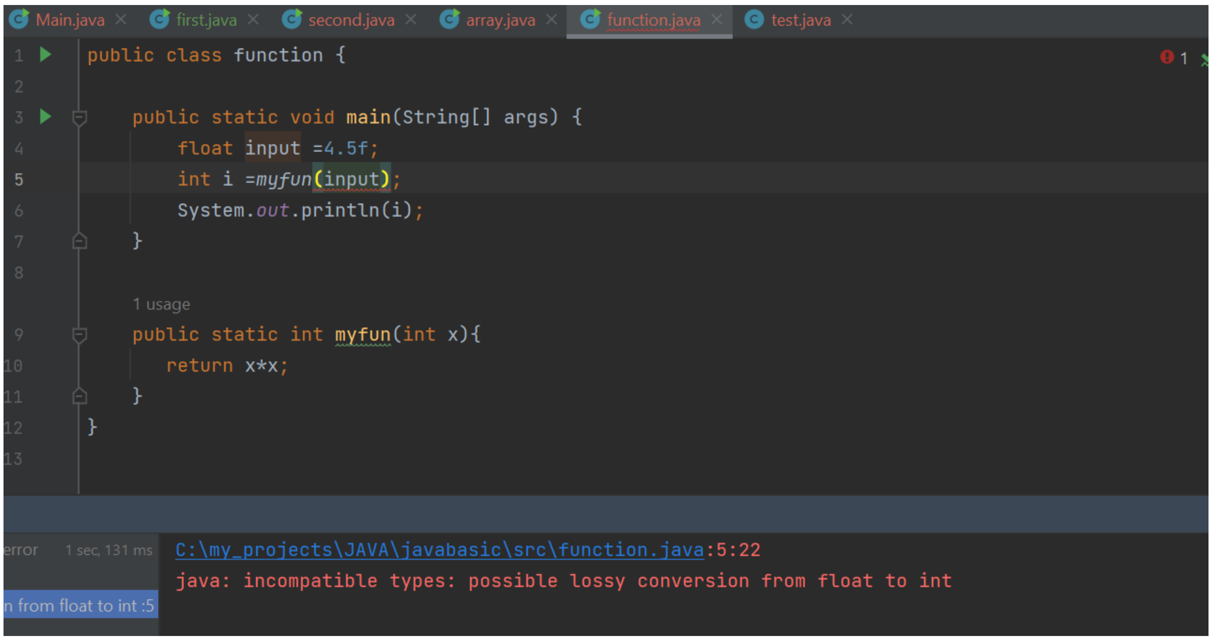
```
double x=1889899.493849384;
float y=4.65f;
```

  We can check the size of any datatype in JAVA - The below statements will print the size of respective datatypes in bits, so If you want the size in bytes then divide it by 8 (because in 1 byte we have 8 bits)

```
System.out.println(Float.SIZE);
System.out.println(Double.SIZE);
System.out.println(Integer.SIZE);
```

- **Method Overloading** - If a class has multiple methods having the same name but different parameters, it is known as Method Overloading. JAVA supports method overloading and for implementing this, we can create multiple functions with the

same name but different parameters. This helps us in applying the function logic to all kinds of data types.
In the below example: We have created a *myfun() function* for doing the *square* of any number, Now if we wanted to do this functionality for a "*float*" input also, Then it will fail, As in our function parameter, we have hard coded that it can take only "*int*" as input. Now to solve this we will do Method overloading.



```
public class function {

    public static void main(String[] args) {
        float input =4.5f;
        int i =myfun(input);
        System.out.println(i);
    }

    1 usage
    public static int myfun(int x){
       return x*x;
    }
}
```

```
error    1 sec, 131 ms    C:\my_projects\JAVA\javabasic\src\function.java:5:22
                          java: incompatible types: possible lossy conversion from float to int
n from float to int :5
```

Now Checkout the below Code, Here we have used Function overloading.

```
public class function {
  public static void main(String[] args) {
      float x1=4.5f;
      int x2=7;
      double x3=454545.67;
/////////Calling function with "int" input//////////////////
      int intOutput =myfun(x2);
      System.out.println(intOutput);
////////////Calling function with float input/////////////////////
      float floatOutput =myfun(x1);
      System.out.println(floatOutput);
////////Calling function with Double input/////////////////////////
      double doubleOutput =myfun(x3);
      System.out.println(doubleOutput);
  }
/////////////Defined Function with different parameters.////
  public static int myfun(int x){return x*x;}
  public static float myfun(float x){return x*x;}
  public static double myfun(double x){return x*x;}

}
```

Output -
49

```
20.25
2.0661176611574887E11
```

Remember - 2 functions with the same name and same signature can't be created, It will not be allowed.

- **Functions** get space for their variables from Stack memory, So whenever a function is called one ***Activation Record*** of that function is been created in **Stack memory**. As we know ***main()*** is the entry point for our JAVA app, So whenever we run a JAVA program automatically an Activation record for main() gets created in Stack memory. Now as different functions have their separate activation record, So if we create a variable with the same name in two functions simultaneously then it will not create any issue, as they are been declared in their private space.
  Variables defined inside functions is a **local variables**.
  As the "**return**" keyword comes, the function returns the data, and the Activation record of that function vanishes out from Stack memory.

- Now as we know about *method overloading*, where we can create multiple functions with the same name but different parameters, So Whenever any of them will be called a separate *activation record* will be created in stack memory.

- Now in *method overloading,* the *return type* of the function didn't matter, So if your function has the same names and same parameters(which means the same datatype and the same number of inputs) then It will through an error, *you can't create two functions with the same name and same parameters*. So as shown below we can not create two functions like this
  ```
  public static int myfun(int x){return x*x;}
  public  static float myfun(int x){return (float) (x*x);}
  ```

  But If we change here function parameters, then it will work fine, as shown below.
  ```
              public static int myfun(int x){return x*x;}
  public  static float myfun(int x,int y){return (float) (x*x);}
  ```

- A loop is a sequence of instructions that is continually repeated until a certain condition is reached. To create a loop in JAVA Check below program,

  This is a simple loop, which will print values from *1 to 10*
  ```
  for (int i=1;i<11;++i){
     System.out.println(i);
  }
  ```

  The "***For***" loop takes 3 statements as input the first statement runs only one time, The second statement looks for a condition and the third condition executes every time the loop runs. We can visualize it better with the below example -
  ```
  int i=1;
  for (System.out.println("Printing--");i<5; System.out.println("Done")){
  ```

```
  System.out.println(i);
  ++i;
}
```

Output -
Printing–
1
Done
2
Done
3
Done
4

- **Traversing** an **Array** in JAVA -

```
int i=0;
String arr[] = {"Rohit","Rahul","Virat", "SKY"};
for (System.out.println("India top order batsman - > "); i<arr.length; ++i ){
 System.out.println(arr[i]);
}
```

Output -
India top order batsman - >
Rohit
Rahul
Virat
SKY

- We have one more interesting and short way to traverse an array.

```
String arr[] = {"Rohit","Rahul","Virat", "SKY"};
for (String name:arr){
  System.out.println(name);
}
```

It will print all the elements of the array. It will go to the array, Pick the first element and store it in variable "***name***" and then print it, now it will pick the second element, print it, and so on.

- Passing an array to a function

```
public class function {
  public static void main(String[] args) {
      int marks[]={55,96,56,0,23,33};
      ChangeMyMarks(marks); //passing Address of array
      for (int i:marks){
          System.out.println(i);
      }
public static void ChangeMyMarks(int score[]){
  score[3]=100;
  }
}
```

Here we are passing the reference of the array to the function, now when the function has the reference, It can directly go to the array and read/write it.

- In JAVA, we have JVM due to which we can’t directly interact with Resources, JVM will interact/manage them on our behalf. So In the above examples, as we have seen how to pass a reference of an array to a function and how to pass variable data to a function, Now both the way of calling function in JAVA we call it as “**Passing Parameter**” Instead of naming them as **Call by reference** and **call by value** respectively.