

Smart Debug Approach

(Know your Errors & Exceptions)

Using Java SE 8©

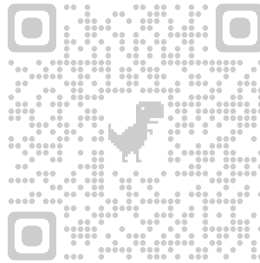
Aniket Singh

Infosys

Test Engineer

LinkedIn

(<https://in.linkedin.com/in/aniket-singh01>)



Overview

A code is an instruction to the system for performing certain defined tasks. However these instructions or tasks are written while adhering to certain laws which are either logical or within bounds of the language (for example 'logical' - could be say if we are in a different country and happen to use the wrong gender to address someone and likewise 'bounds' would be if we mess up the grammar to form a sentence). Now coming to Java, this document tends to gear up your vision or outlook of debugging any code in a glance for issues, if present.

Types of Codes:

1. Good Code:
 - a. Optimised code (When execution and memory allocations are only spent exactly the much is needed, also lesser **Time Complexity** is followed)
2. Bad Code:
 - a. It runs without any interruption but surely has room for improvement in terms of memory allocation or improvement in **Time Complexity**.
 - b. Has a potential to interrupt execution or either result in abnormal or unexpected Output

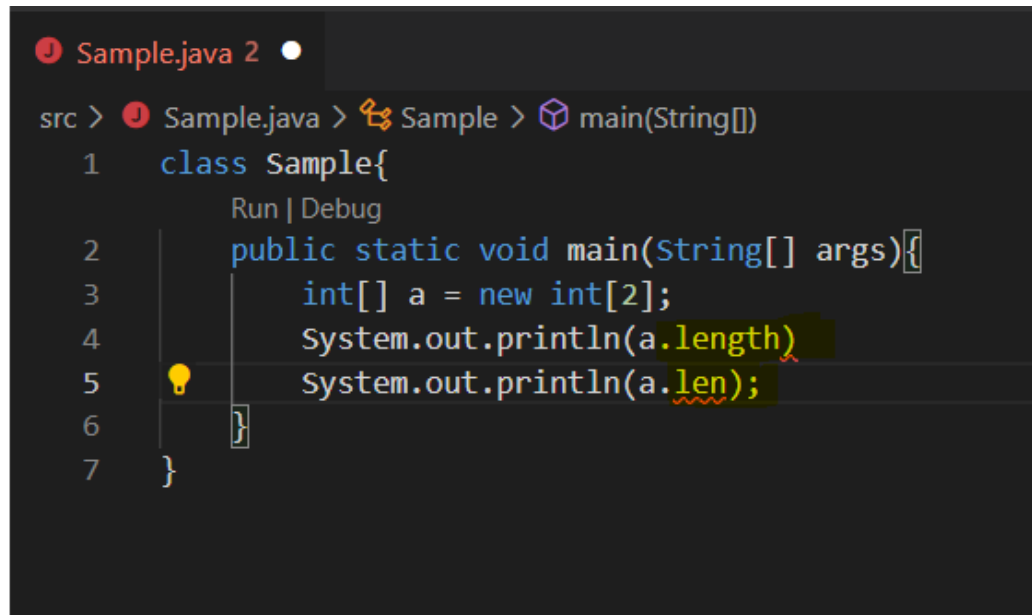
Lets focus on Bad code

It is very important to understand to what should not be done while writing a code in any language, while speaking for Java, we should remember the below elements which could bring any code at halt:

- Compile-time (**Errors**)
- Run-Time (**Exceptions**)
- Logical (**Errors | Exceptions**) - Sometimes code runs in this case only because the programmer was asked to do Task 'A' but he ends up doing a different task, this can happen if the programmer has inadequate idea/concept for implementing the solution.

Compile Time errors

These are the result of performing illegal operations or writing code which will basically prevent code to run - this is primarily the case when Syntax is wrong (missing a semicolon, missing a closing curly brace etc) . Let's look into this closely.

A screenshot of an IDE window titled 'Sample.java 2'. The code is as follows:

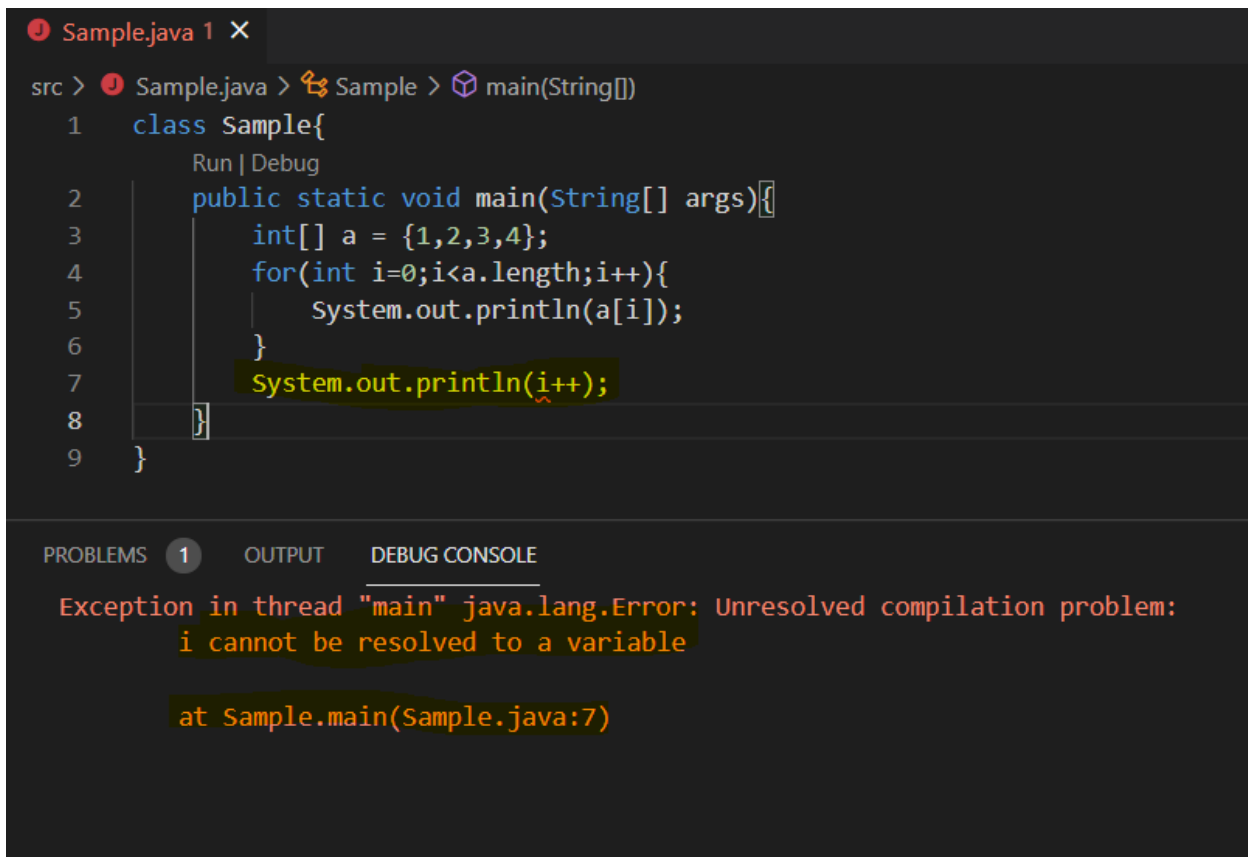
```
src > Sample.java > Sample > main(String[])
1  class Sample{
    Run | Debug
2      public static void main(String[] args){
3          int[] a = new int[2];
4          System.out.println(a.length)
5          System.out.println(a.len);
6      }
7  }
```

Line 4 has a yellow lightbulb icon next to it, indicating a warning or error. Line 5 has a red squiggly line under the word 'len', indicating an error. The code is otherwise syntactically correct.

Line 4 - Has a missing semicolon.

Line 5 - Has an incorrect reference to array class object 'a'. In Arrays, there is no property called 'len' however there is one called 'length' which basically tells how many elements the array can accommodate.

Let's look at some complicated examples:



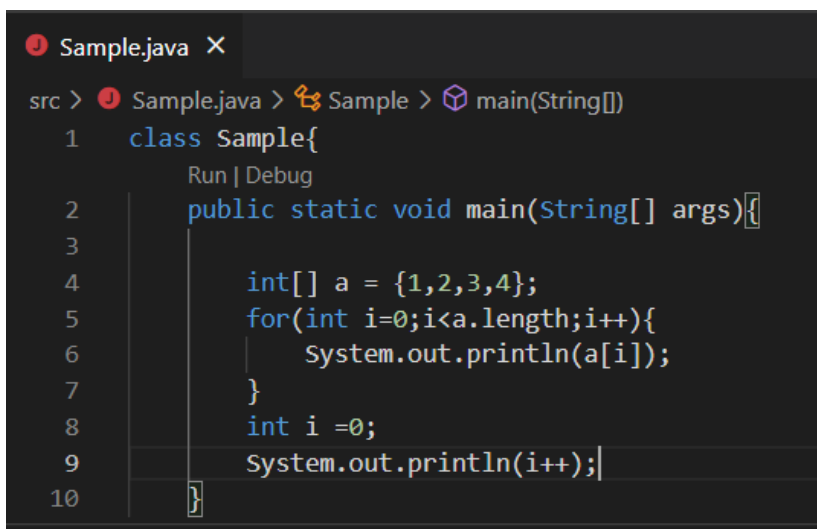
```
Sample.java 1 X
src > Sample.java > Sample > main(String[])
1 class Sample{
    Run | Debug
2     public static void main(String[] args){
3         int[] a = {1,2,3,4};
4         for(int i=0;i<a.length;i++){
5             System.out.println(a[i]);
6         }
7         System.out.println(i++);
8     }
9 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
i cannot be resolved to a variable
at Sample.main(Sample.java:7)

Line 7 here in the above image is resulting in an error because 'i' is local to only that 'for' loop and outside the loop 'i' means nothing to the compiler.

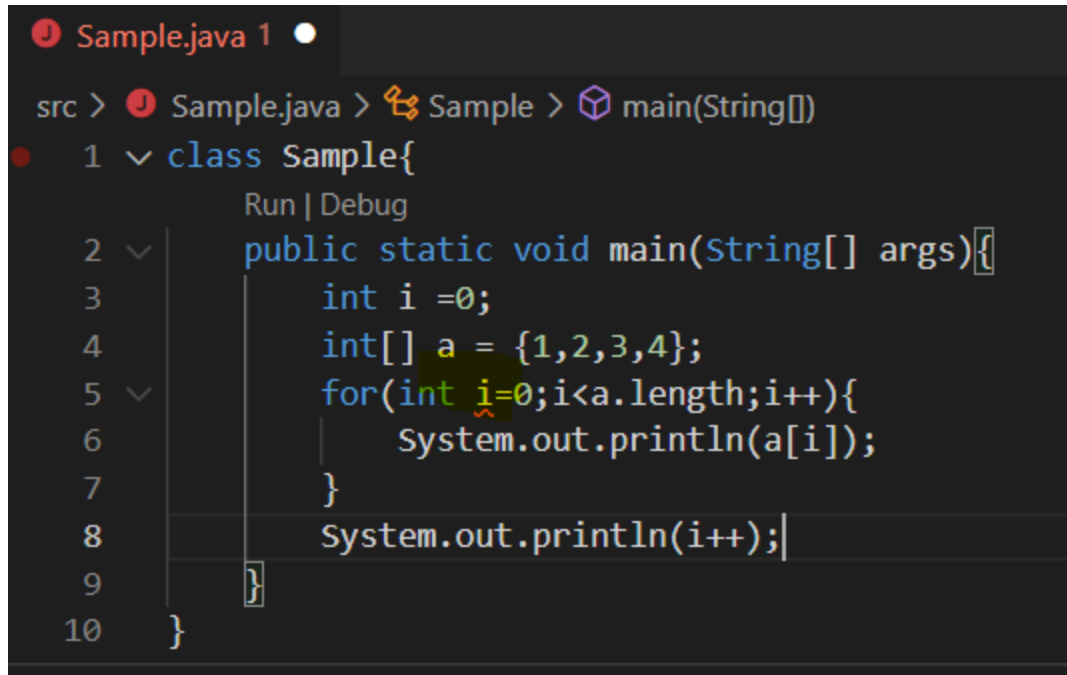
However, when I modify the same code:



```
Sample.java X
src > Sample.java > Sample > main(String[])
1 class Sample{
    Run | Debug
2     public static void main(String[] args){
3
4         int[] a = {1,2,3,4};
5         for(int i=0;i<a.length;i++){
6             System.out.println(a[i]);
7         }
8         int i = 0;
9         System.out.println(i++);
10    }
```

The error is no longer present (no red line), because 'i' is now also a variable which is local to the outside block (things present outside the 'for' loop).

But if I would have written the 'i' before the line of 'for' loop, it would be an issue.



```

Sample.java 1
src > Sample.java > Sample > main(String[])
1  class Sample{
    Run | Debug
2  public static void main(String[] args){
3      int i =0;
4      int[] a = {1,2,3,4};
5      for(int i=0;i<a.length;i++){
6          System.out.println(a[i]);
7      }
8      System.out.println(i++);
9  }
10 }
```

Now one must say, that if I have already declared 'i' then why am I seeing a red line - meaning error in code. Well that's because the same declaration has happened twice here. When you declare a variable, you mention their data type, but when you initialize or modify them, then you don't need to write 'int' or the relevant data type again and again.

Like -

1. `Int i = 0;` //Says that I declare and initialize value at same time
2. `Int i;` //Is just a variable of data type int, thus reserving 4 bytes of memory in your system
3. `I = 6;` // Now i is holding a value of 6 (certain number of bits used from 4 bytes of memory reserved)

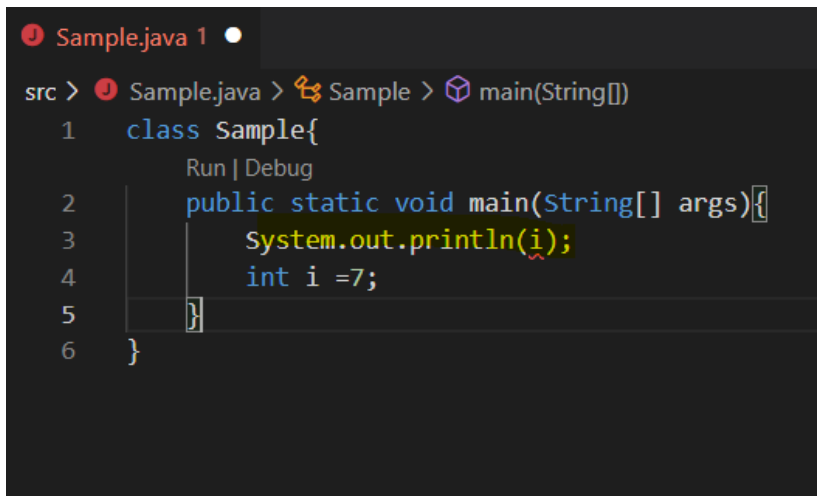
Now point 2 and 3 can be used in the same sequence as one is declaring, another is updating the value, but not point 1 and 2 one after the other as both declare the same data type using the same variable name.

However,

1. `int i;`
2. `float i;`

The above 2 points are illegal too if used one after the other, always remember - **Variable names should be unique in the same code block, no matter if they refer to different data types.**

Another set of examples:

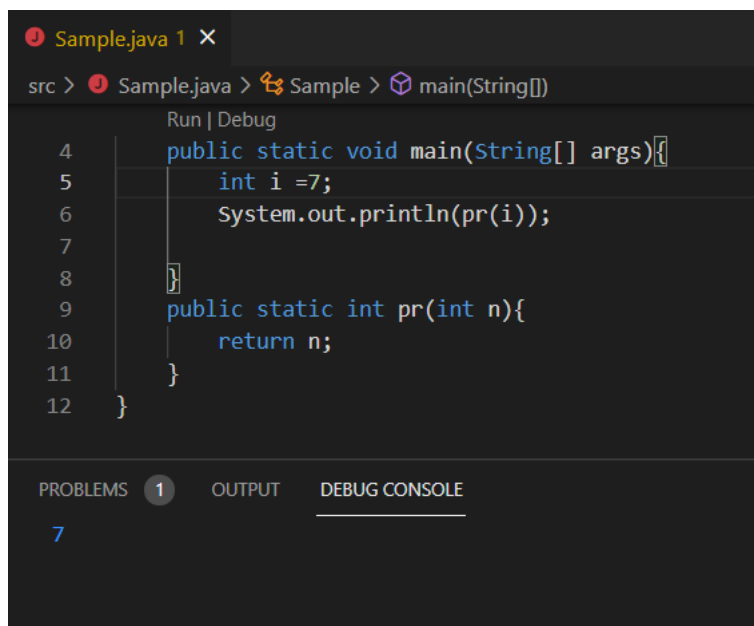


```
Sample.java 1
src > Sample.java > Sample > main(String[])
1 class Sample{
2     public static void main(String[] args){
3         System.out.println(i);
4         int i =7;
5     }
6 }
```

Line 3 - print statement throws an error.

Reason - 'i' was being used before its 'declarative' line - Any variable cannot be used if its not declared above the particular operation line(here in this case is print statement)

However same is not true with methods: They can be used even if they are defined below within the same class



```

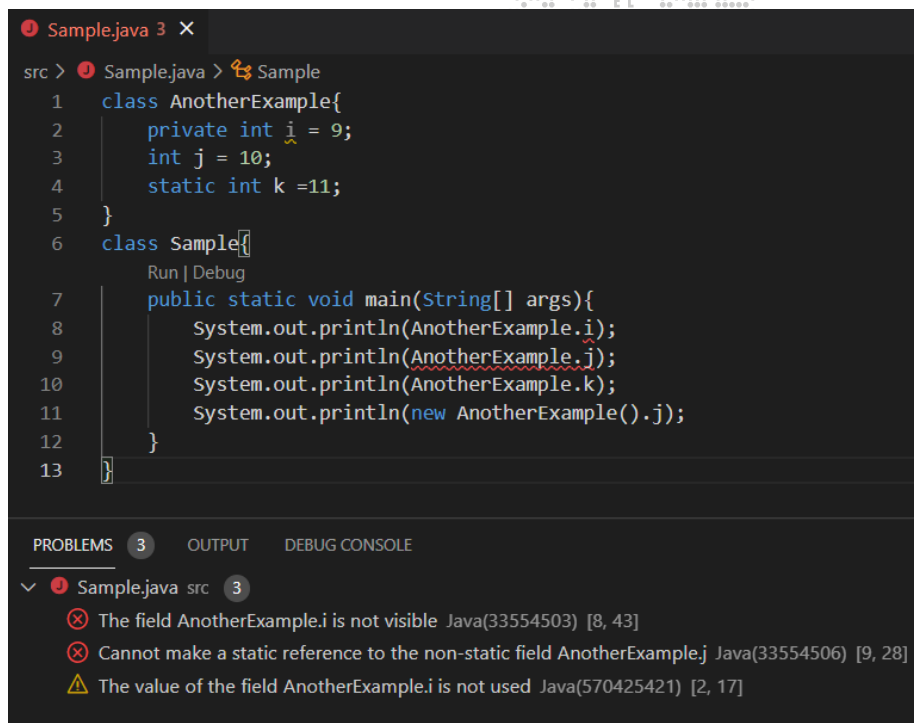
Sample.java 1 X
src > Sample.java > Sample > main(String[])
Run | Debug
4 public static void main(String[] args){
5     int i =7;
6     System.out.println(pr(i));
7
8 }
9 public static int pr(int n){
10     return n;
11 }
12 }

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

7

Now let's have a look into examples where we directly oppose the concepts of Java or OOPS



```

Sample.java 3 X
src > Sample.java > Sample
1 class AnotherExample{
2     private int i = 9;
3     int j = 10;
4     static int k =11;
5 }
6 class Sample{
7     Run | Debug
8     public static void main(String[] args){
9         System.out.println(AnotherExample.i);
10        System.out.println(AnotherExample.j);
11        System.out.println(AnotherExample.k);
12        System.out.println(new AnotherExample().j);
13    }
14 }

```

PROBLEMS 3 OUTPUT DEBUG CONSOLE

Sample.java src 3

- ✗ The field AnotherExample.i is not visible Java(33554503) [8, 43]
- ✗ Cannot make a static reference to the non-static field AnotherExample.j Java(33554506) [9, 28]
- ⚠ The value of the field AnotherExample.i is not used Java(570425421) [2, 17]

PROBLEMS 3 OUTPUT DEBUG CONSOLE

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    The field AnotherExample.i is not visible
    Cannot make a static reference to the non-static field AnotherExample.j

    at Sample.main(Sample.java:8)
```

In the above code and its output,

Line 8 is trying to access 'i' which is private to Class 'AnotherExample' only and as we know from OOPs that private entities cannot be accessed outside its class. Hence its not visible from outside

Line 9 is trying to access 'j' which is non-static in nature - Now always remember, everything outside main() method in a class should be declared as static to be used in main method, since the whole context is static, even if you look at - public **static** void main(String[] args) is static in nature, elements within it are not however

Line 10, does not throw an error since 'k' is already static in nature and thus the static reference is ok here.

Also, just so you know -

A static reference is when you directly access a class member, where as for non-static members you need to create an object of the class then access the member via the object.

Class A{

int x =9;

}

Class B{

public static void main(String[] args){

A a = new A();

System.out.println(a.x); //You create an object 'a' of type 'A' and you access 'int x'

through 'a'.

}

}

Line 11 is doing the same thing I just explained right now, but in a short way. Even "new A()" is in itself saying "I'm an object which has invoked the default constructor."

Now let's get into the final gear of Compile time errors - the examples that you'll notice now is when things are wrong with Interfaces, Abstract classes.

Below is an example of possible things that could go wrong in Interface:

```

Sample.java 5 x
src > Sample.java > Sample > main(String[])
1  interface AnotherExample{
2      int j = 10;
3      static int k =11;
4      public abstract void f1();
5      private int i = 9;
6      public abstract void f2(){
7
8      }
9      public static abstract void func();
10 }
11 //Remember diamond problem
12 class Sample implements AnotherExample{
13     public static void main(String[] args){
14         System.out.println(j);
15         System.out.println(k);
16     }
17 }

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE Filter (e.g.)

- ✗ Illegal modifier for the interface field AnotherExample.i; only public, static & final are permitted Java(33554775) [5, 17]
- ✗ Abstract methods do not specify a body Java(603979889) [6, 26]
- ✗ Illegal combination of modifiers for the interface method func; only one of abstract, default, or static permitted Java(67109920) [9, 33]
- ✗ The type Sample must implement the inherited abstract method AnotherExample.f2() Java(67109264) [12, 7]
- ✗ The type Sample must implement the inherited abstract method AnotherExample.f1() Java(67109264) [12, 7]

Bunch of issues right?

Line 5: In an Interface only public, static or final is permitted to be used - A good question could be why only these 3? What happens to Private and Protected members? Well if you read carefully we are not talking about a 'Class' but an 'Interface' and why do I stress on this?

Let's go back to OOPS principle:

A class encapsulates all members within a logical boundary

Whereas an 'Interface' if we go by its literal meaning - something we can interact with or use again and again, like a template, so my question would be here is, do you need something to be hidden in a template, which will be used or implemented in so many different ways?

I have made the answer obvious so I will not state the fact for obvious.

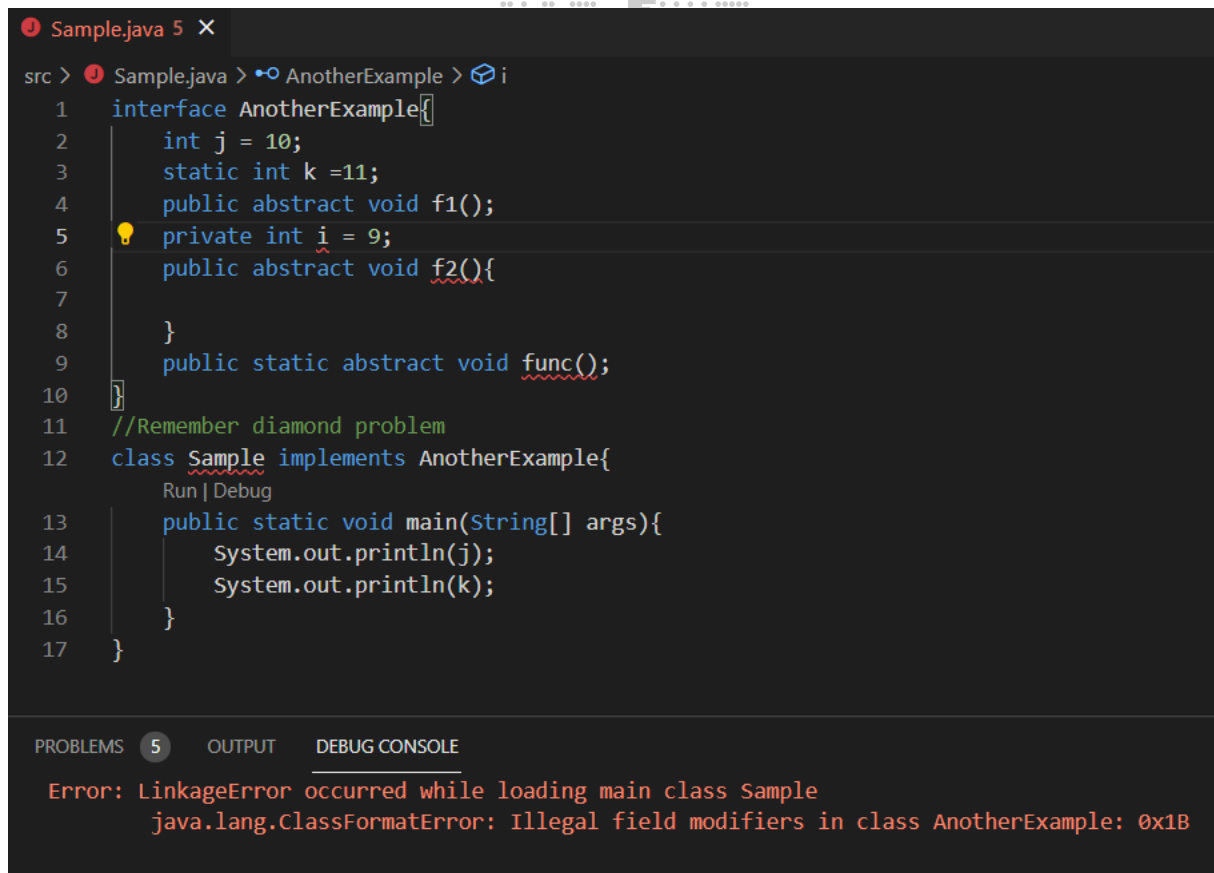
But if you still need to ask? Think of a Resume template found online to build your CV.

Line 6: An abstract method is like a blueprint. So it does not have any braces to contain the code, it's meant to be overridden in a class where the interface containing the abstract method is implemented. *Also remember an Abstract method can be defined in Abstract class or Interface only.*

Line 9: Two modifiers in one statement 'static' and 'abstract'.

Line 12: When an interface is implemented by a Class, its necessary for the implementing class to implement its abstract methods (basically override them - I'll show this in later part)

Now let's look at the results on compiling the code with issues, just to familiarize ourselves with Errors



```

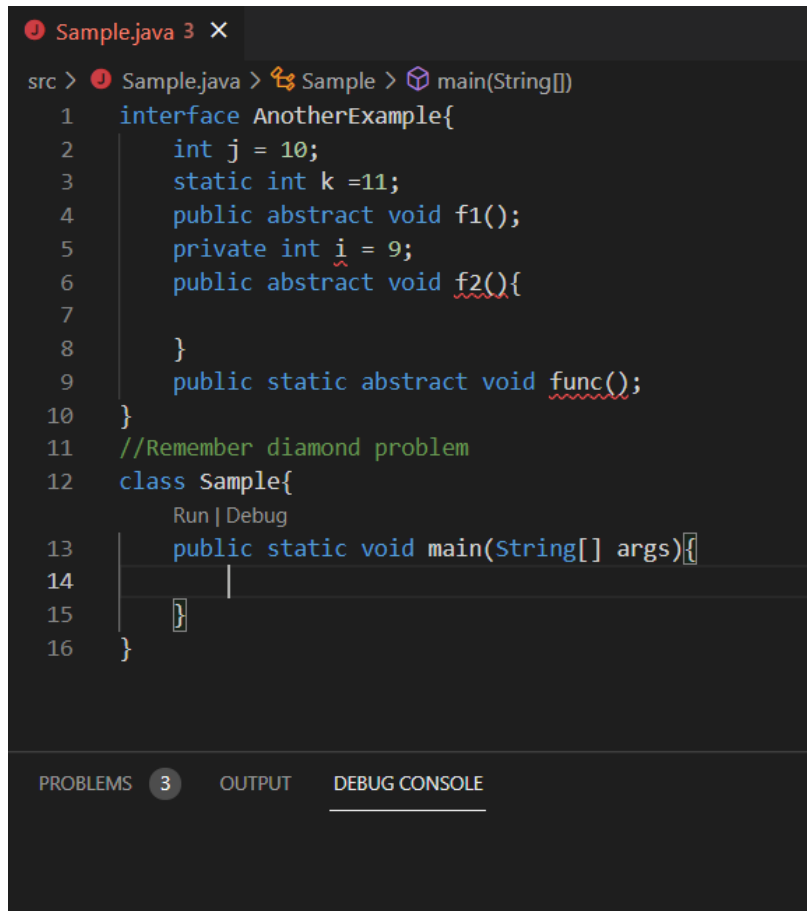
Sample.java 5 X
src > Sample.java > AnotherExample > i
1  interface AnotherExample{
2      int j = 10;
3      static int k =11;
4      public abstract void f1();
5      private int i = 9;
6      public abstract void f2(){
7
8      }
9      public static abstract void func();
10 }
11 //Remember diamond problem
12 class Sample implements AnotherExample{
13     public static void main(String[] args){
14         System.out.println(j);
15         System.out.println(k);
16     }
17 }

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE

Error: LinkageError occurred while loading main class Sample
 java.lang.ClassFormatError: Illegal field modifiers in class AnotherExample: 0x1B

Below is when I don't implement this Interface having issues



```
Sample.java 3 X
src > Sample.java > Sample > main(String[])
1 interface AnotherExample{
2     int j = 10;
3     static int k =11;
4     public abstract void f1();
5     private int i = 9;
6     public abstract void f2(){
7
8     }
9     public static abstract void func();
10 }
11 //Remember diamond problem
12 class Sample{
13     Run | Debug
14     public static void main(String[] args){
15
16     }
17 }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE

No output

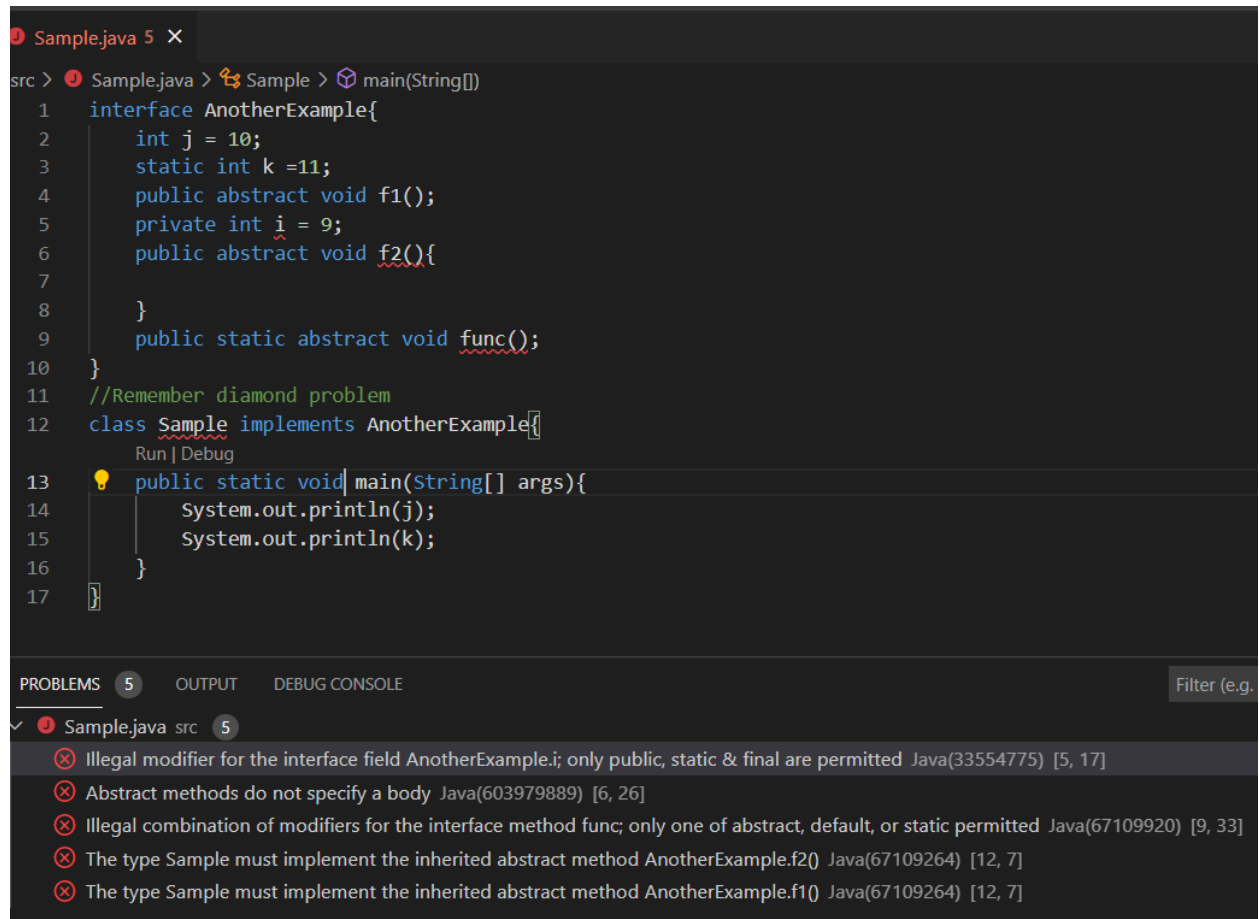
Reason: class Sample was compiled since it has nothing in it which could cause an error or even print anything.

Why did the Interface not throw any error?

- 1 - Not a class.
- 2- No main method.
- 3 - Not implemented in the executing class.

Now let's go back to the original code with issues fixed.

Before:



```
Sample.java 5 X
src > Sample.java > Sample > main(String[])
1 interface AnotherExample{
2     int j = 10;
3     static int k =11;
4     public abstract void f1();
5     private int i = 9;
6     public abstract void f2(){
7
8     }
9     public static abstract void func();
10 }
11 //Remember diamond problem
12 class Sample implements AnotherExample{
13     public static void main(String[] args){
14         System.out.println(j);
15         System.out.println(k);
16     }
17 }
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE Filter (e.g.)

- ✗ Illegal modifier for the interface field AnotherExample.i; only public, static & final are permitted Java(33554775) [5, 17]
- ✗ Abstract methods do not specify a body Java(603979889) [6, 26]
- ✗ Illegal combination of modifiers for the interface method func; only one of abstract, default, or static permitted Java(67109920) [9, 33]
- ✗ The type Sample must implement the inherited abstract method AnotherExample.f2() Java(67109264) [12, 7]
- ✗ The type Sample must implement the inherited abstract method AnotherExample.f1() Java(67109264) [12, 7]

After:

```
rc > Sample.java > AnotherExample
1  interface AnotherExample{
2      int j = 10;
3      static int k =11;
4      public abstract void f1();
5      int i = 9;
6      public abstract void f2();
7      public abstract void func();
8  }
9  class Sample implements AnotherExample{
    Run | Debug
10     public static void main(String[] args){
11         System.out.println(j);
12         System.out.println(k);
13     }
14
15     public void f1(){
16         System.out.println("f1 is overridden");
17     }
18     public void f2(){
19         System.out.println("f2 is overridden");
20     }
21     public void func(){
22         System.out.println("func is overridden");
23     }
24 }
25 }
```

Output:

PROBLEMS	OUTPUT	DEBUG CONSOLE
	10	
	11	

Values of 'j' and 'k' are printed, we can even print those functions which we overrode.

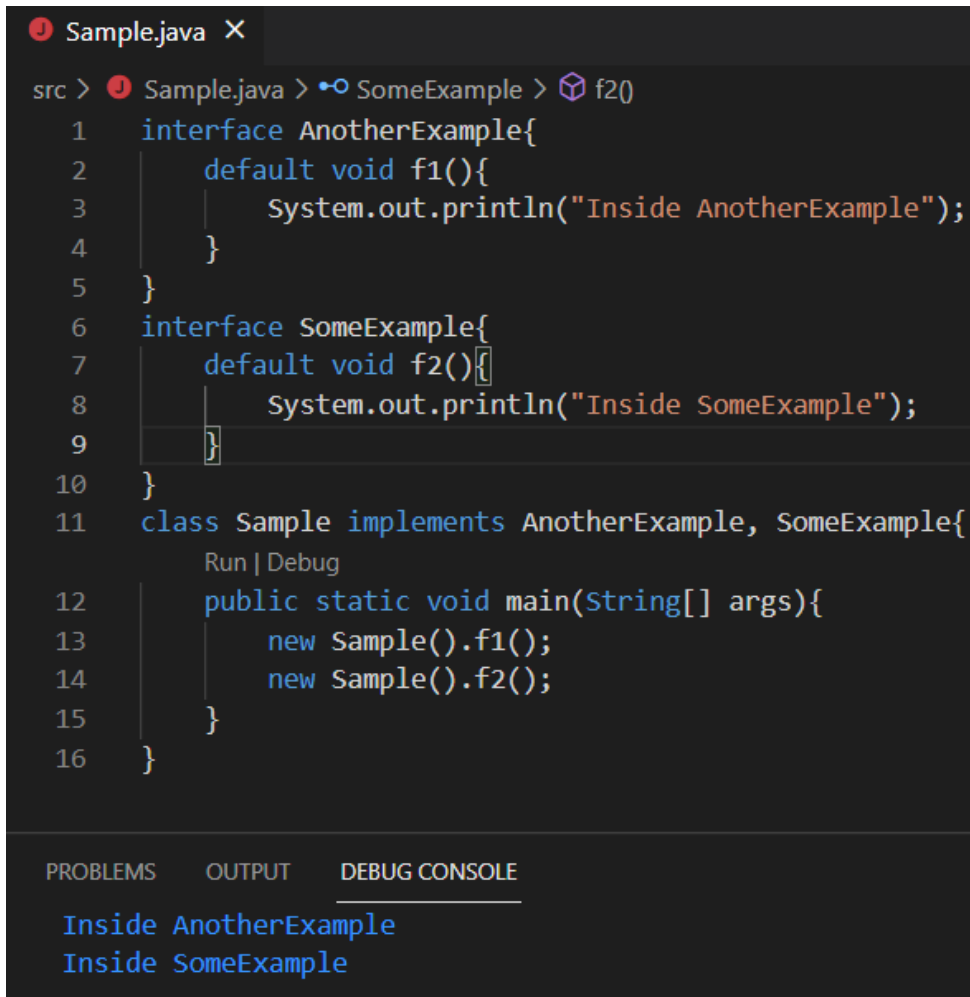
```
10 class Sample implements AnotherExample{
    Run | Debug
11     public static void main(String[] args){
12         System.out.println(j);
13         System.out.println(k);
14         new Sample().f1();
15         new Sample().f2();
16         new Sample().func();
17     }
18 }
19
20     public void f1(){
21         System.out.println("f1 is overridden");
22     }
23     public void f2(){
24         System.out.println("f2 is overridden");
25     }
26     public void func(){
27         System.out.println("func is overridden");
28     }
29 }
```

PROBLEMS OUTPUT DEBUG CONSOLE

```
10
11
f1 is overridden
f2 is overridden
func is overridden
```



Now let me show you a code with 'default' methods in interfaces



```
Sample.java X
src > Sample.java > SomeExample > f2()
1 interface AnotherExample{
2     default void f1(){
3         System.out.println("Inside AnotherExample");
4     }
5 }
6 interface SomeExample{
7     default void f2(){
8         System.out.println("Inside SomeExample");
9     }
10 }
11 class Sample implements AnotherExample, SomeExample{
12     public static void main(String[] args){
13         new Sample().f1();
14         new Sample().f2();
15     }
16 }
```

Run | Debug

PROBLEMS OUTPUT DEBUG CONSOLE

Inside AnotherExample
Inside SomeExample

No issues right?

Let's mess it up and look at an issue around ambiguity when two interfaces are implemented by one class and both have the same function name.



```

Sample.java 1 X
src > Sample.java > SomeExample > f1()
1 interface AnotherExample{
2     default void f1(){
3         System.out.println("Inside AnotherExample");
4     }
5 }
6 interface SomeExample{
7     default void f1(){
8         System.out.println("Inside SomeExample");
9     }
10 }
11 class Sample implements AnotherExample, SomeExample{
12     public static void main(String[] args){
13         new Sample().f1();
14     }
15 }

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

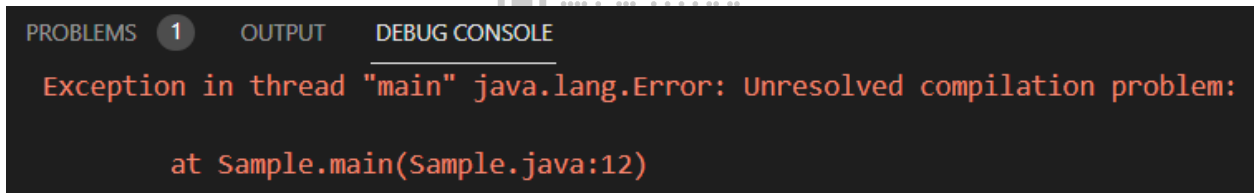
Sample.java src 1

✗ Duplicate default methods named f1 with the parameters () and () are inherited from the types SomeExample and AnotherExample Java(67109917) [11, 7]

Self explanatory?

No? Reason - Compiler is seeing ambiguity as to whom it should refer when 'Sample' class object is created and f1() is called.

Also the error thrown:

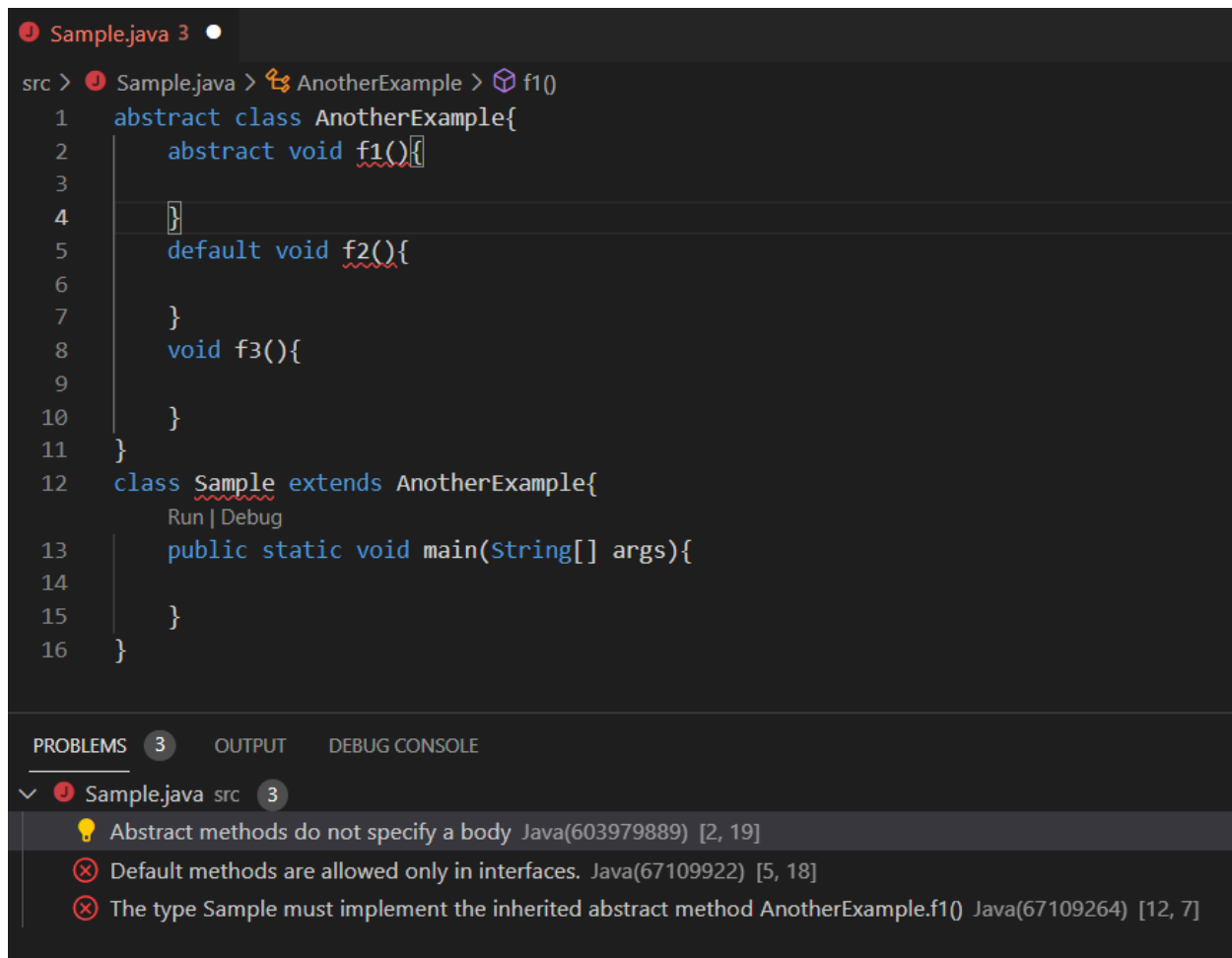


PROBLEMS 1 OUTPUT DEBUG CONSOLE

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

at Sample.main(Sample.java:12)

Now let's look at some examples related to Abstract classes:



The screenshot shows an IDE window titled 'Sample.java 3'. The code is as follows:

```
src > Sample.java > AnotherExample > f1()
1  abstract class AnotherExample{
2      abstract void f1(){
3
4      }
5      default void f2(){
6
7      }
8      void f3(){
9
10     }
11 }
12 class Sample extends AnotherExample{
13     public static void main(String[] args){
14
15     }
16 }
```

Below the code editor, the 'PROBLEMS' tab is active, showing three errors:

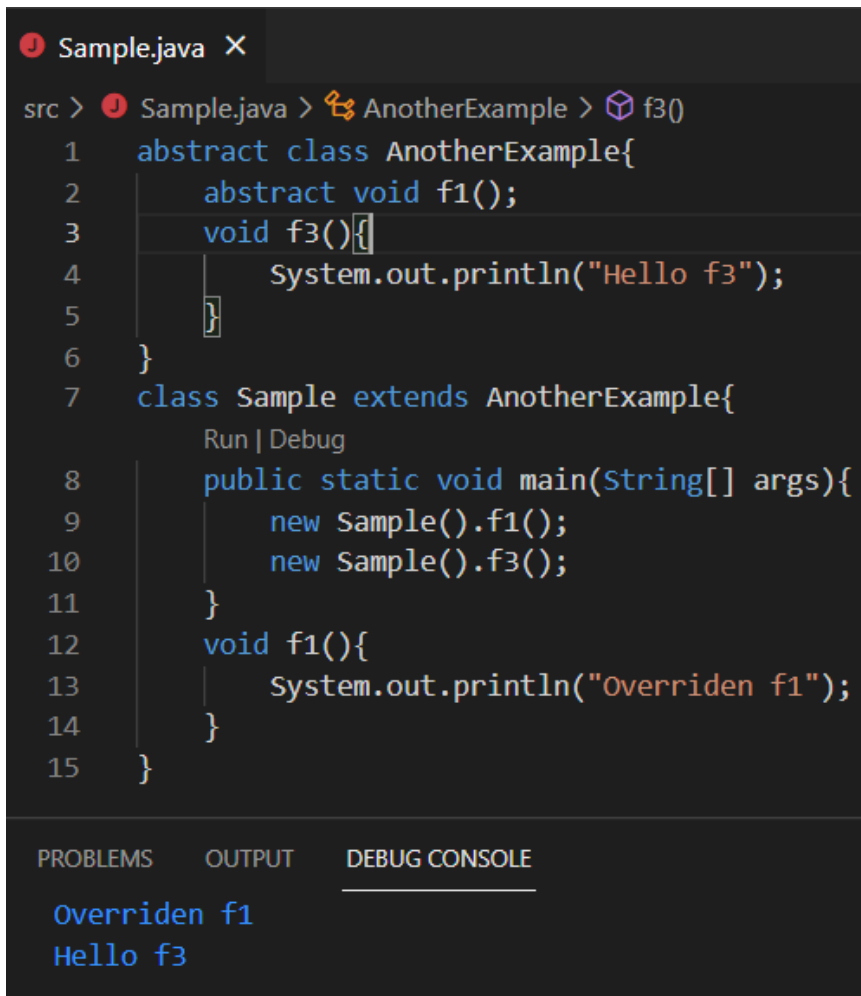
- Abstract methods do not specify a body Java(603979889) [2, 19]
- Default methods are allowed only in interfaces. Java(67109922) [5, 18]
- The type Sample must implement the inherited abstract method AnotherExample.f1() Java(67109264) [12, 7]

By now, you would have understand what is causing error for Line 2, if not then you can scroll back up on Interface part where I have explained about abstract methods

Line 5: Only interface is allowed to have 'default' methods, since it is something like a template, it can't be private or protected so it really makes sense that why Class or Abstract class will not have default methods

Line 12: It is asking the class Sample to override the unimplemented methods from the extended abstract class.

Now I'll fix the code and we can observe the result.



```
Sample.java X
src > Sample.java > AnotherExample > f3()
1  abstract class AnotherExample{
2      abstract void f1();
3      void f3(){
4          System.out.println("Hello f3");
5      }
6  }
7  class Sample extends AnotherExample{
8      public static void main(String[] args){
9          new Sample().f1();
10         new Sample().f3();
11     }
12     void f1(){
13         System.out.println("Overriden f1");
14     }
15 }
```

Run | Debug

PROBLEMS OUTPUT DEBUG CONSOLE

Overriden f1
Hello f3

Method 'f1()' is rectified in class 'AnotherExample' by removing the braces and placing the semicolon, also it's implemented in the Sample class which had extended 'AnotherExample'. Lastly 'default' method was removed following the OOPs principles for Classes.

Run Time errors

These are the result of performing certain operations which can lead to abnormal execution or interrupt the workflow. However these kinds of errors do not prevent the code to run and interruptions can be handled sophisticatedly using 'try-catch' block. These are mostly 'Exceptions'.

Let's look at a few examples.

```
Sample.java 2 X
src > Sample.java > Sample > main(String[])
1  class Sample{
    Run | Debug
2      public static void main(String[] args){
3          String s = "Hello";
4          int[] a = null;
5          try {
6              System.out.println(5/0);
7          } catch (Exception e) {
8              e.printStackTrace();
9          }
10         try {
11             System.out.println(args[2]);
12         } catch (Exception e) {
13             e.printStackTrace();
14         }
15         try {
16             System.out.println(s.charAt(10));
17         } catch (Exception e) {
18             e.printStackTrace();
19         }
20         try {
21             int i = Integer.parseInt(s);
22         } catch (Exception e) {
23             e.printStackTrace();
24         }
25         try {
26             System.out.println(a.length);
27         } catch (Exception e) {
28             e.printStackTrace();
29         }
30     }
31 }
```

Let's look at the result on compiling this.

```

java.lang.ArithmeticException: / by zero
    at Sample.main(Sample.java:6)                                     Sample.java:6
java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 0
    at Sample.main(Sample.java:11)                                   Sample.java:11
java.lang.StringIndexOutOfBoundsException: String index out of range: 10
    at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)   StringLatin1.java:48
    at java.base/java.lang.String.charAt(String.java:709)              String.java:709
    at Sample.main(Sample.java:16)                                     Sample.java:16
java.lang.NumberFormatException: For input string: "Hello"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)   NumberFormatException.java:68
    at java.base/java.lang.Integer.parseInt(Integer.java:658)          Integer.java:658
    at java.base/java.lang.Integer.parseInt(Integer.java:776)          Integer.java:776
    at Sample.main(Sample.java:21)                                     Sample.java:21
java.lang.NullPointerException
    at Sample.main(Sample.java:26)                                     Sample.java:26

```

Line 6 : we know any number divide by 0 is not logical

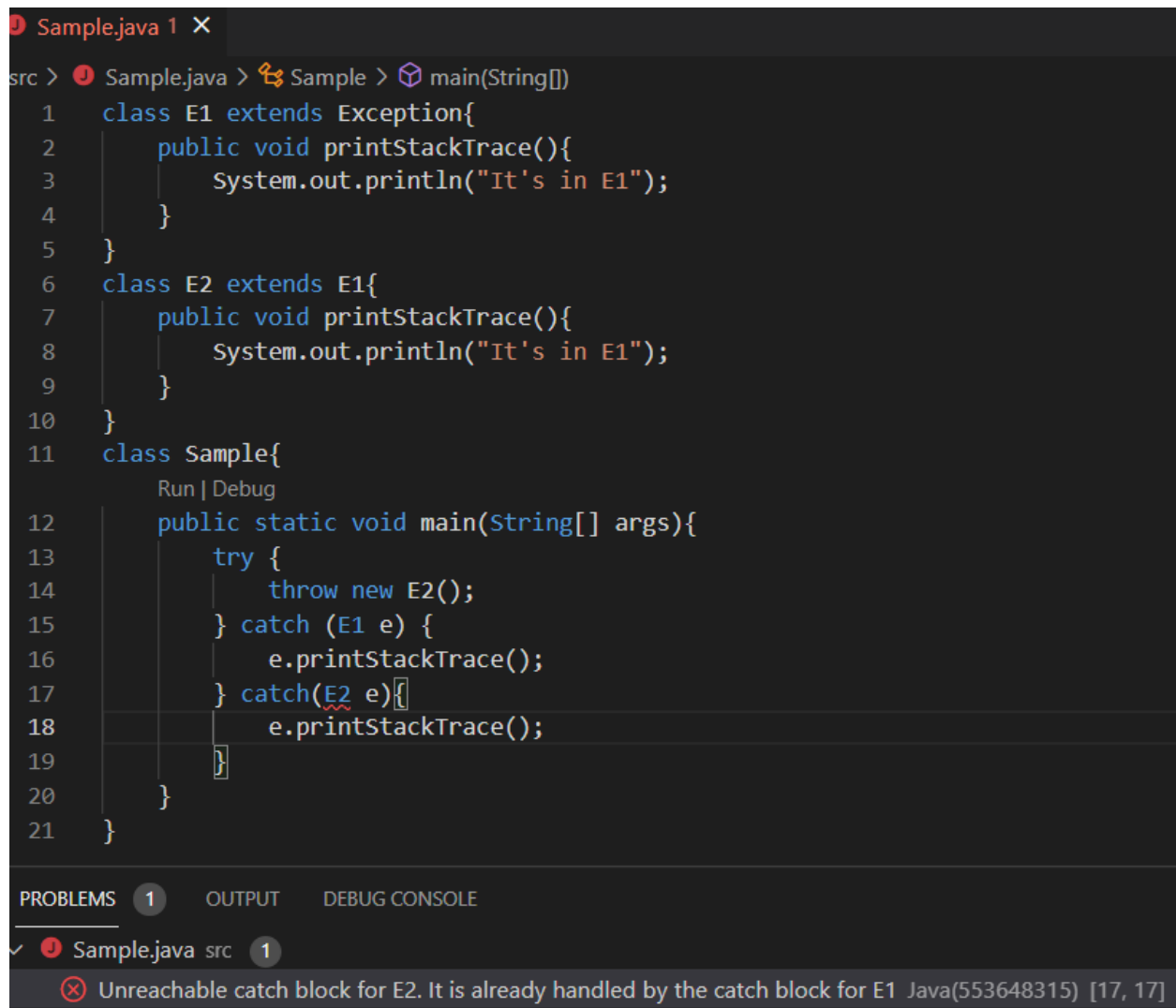
Line 11: 'args' array is the same array which is used in public static void main(String[] args) but here it threw an exception since nothing was provided to this array so basically an empty array means no indexes. *'Args' except values only on command line execution - for example if this .java file would have been executed in Command Prompt/Terminal.*

Line 16: The String 's' has only 5 characters but this statement is trying to access a character on the 10th index position of the string, which is not logical.

Line 21: "Hello" is not an Integer that can be converted/parsed as a number, however "2" or "123" can be parsed.

Line 26: Array 'a' is assigned to a null object so now it basically tells us that any property associated with 'a' is not there anymore, since we overrode it with 'null'. If you assign null to any object or variable, you render their properties useless. And here in this case the code is trying to access the property 'length' which on a normally defined array would just return the length of the array but now since it's a null object, no further point.

Now let's look at a complicated example for 'Exceptions'



```
src > Sample.java > Sample > main(String[])
1  class E1 extends Exception{
2      public void printStackTrace(){
3          System.out.println("It's in E1");
4      }
5  }
6  class E2 extends E1{
7      public void printStackTrace(){
8          System.out.println("It's in E1");
9      }
10 }
11 class Sample{
12     Run | Debug
13     public static void main(String[] args){
14         try {
15             throw new E2();
16         } catch (E1 e) {
17             e.printStackTrace();
18         } catch(E2 e){
19             e.printStackTrace();
20         }
21     }
22 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

✓ Sample.java src 1

✗ Unreachable catch block for E2. It is already handled by the catch block for E1 Java(553648315) [17, 17]

Line 17: Order is very much important when dealing with exceptions, now if you look at Class E1 and Class E2. Both are Exception classes

E1 has inherited the Exception class, basically E1 has become a part of the Exception family, also I have overridden the 'printStackTrace' method which is found in Exception class. Also E2 is inherited by E1, practically E1 is parent to E2. So the error we see that Line 17 is unreachable, since the parent exception class (E1) has already handled it within the first catch block that the child was not needed.

Observe the below error on compiling this code

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Unreachable catch block for E2. It is already handled by the catch block for E1
    at Sample.main(Sample.java:17)
```

Now what would happen if catch blocks were rearranged as below:

```
Sample.java 1 X
src > Sample.java > Sample > main(String[])
1  class E1 extends Exception{
2      public void printStackTrace(){
3          System.out.println("It's in E1");
4      }
5  }
6  class E2 extends E1{
7      public void printStackTrace(){
8          System.out.println("It's in E2");
9      }
10 }
11 class Sample{
12     Run | Debug
13     public static void main(String[] args){
14         try {
15             throw new E2();
16         } catch (E2 e) {
17             e.printStackTrace();
18         } catch (E1 e){
19             e.printStackTrace();
20         }
21     }
22 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE

It's in E2

It gives the output from E2 class, now logically this is incorrect but in compilation there was no issue because E1 is a parent and it can be a parent to many such exception classes, so

compiler leaves the possibility of this scenario and does not throw any error, but rather handle error via E2 catch block.

Logical errors

These are similar to 'Exceptions' but there's a scenario where a Programmer might end up doing a different task which is logically different than what was being asked, so including this scenario it also makes a Logical error. Sometimes an 'infinite loop' leading to a crash while throwing an exception of StackOverflow can also be a logical error or even using 'bad -logic' for 'if-else' or 'switch case' can also be considered, which leads to an unoptimized code.

