Correction Advanced Java Class Design - 0

B - class B

D - final class B

Method-local inner classes cannot be defined using explicit access modifiers (public, protected and private) but non-access modifiers: final and abstract can be used with method-local inner class.

In this case, abstract is also not possible as new B() is used.

Correction Advanced Java Class Design - 1

D - Not possible to define the purpose

Lambda expression doesn't work without target type and target type must be a functional interface.

In this case as the given lambda expression is not assigned to any target type, hence its purpose is not clear.

In fact, given lambda expression causes compilation error without its target type.

Correction Advanced Java Class Design - 2

B -

```
200
100
```

Keyword this within anonymous inner class code refers to the instance of anonymous inner class itself, so this.i in anonymous inner class code is 200.

Whereas, keyword this within lambda expression refers to the instance of enclosing class where lambda expression is written, so this.i in lambda expression is 100.

Correction Advanced Java Class Design - 3

D - Dog eats biscuits

A class can have multiple static nested classes. static nested class can use all 4 access modifiers (public, protected, default and private) and 2 non-access modifiers (final and abstract). No issues at Line 2.

static nested class can extend from a class and can implement multiple interfaces so Line 6 compiles fine. No overriding rules were broken while overriding eat() method, so no issues at Line 7.

Test class is outside the boundary of class Outer. So Animal can be referred by Outer.Animal and Dog can be referred by Outer.Dog.

Polymorphism is working in this case, super class (Outer.Animal) reference variable is referring to the instance of sub class (Outer.Dog). So, no issues at Line 15 as well.

Test class compiles and executes successfully and prints “Dog eats biscuits” on to the console.

Correction Advanced Java Class Design - 4

A -

```
Flyable flyable = new Flyable() {
  public void fly() {
    System.out.println( Flying high');
  }
};
```

new Flyable(); => Can't instantiate an interface.

new Flyable(){}; => fly() method is not implemented.

new Flyable() { public void fly() { System.out.println("Flying high"); } } => semicolon missing at the end

new Flyable() { public void fly() { System.out.println("Flying high"); } }; => correct syntax

Correction Advanced Java Class Design - 5

B - Compilation error

At the time of creating the instance of anonymous inner class, new Shape() is used, which means it is looking for a no-argument constructor in anonymous inner class code, which would invoke the no-argument constructor of super class, Shape.

But as parameterized constructor is specified in Shape class, so no-argument constructor is not provided by the compiler and hence compilation error.

To correct the compilation error, pass the enum constant while instantiating anonymous inner class.

Shape shape = new Shape(ShapeType.CIRCLE) {...}; or you can even pass null: Shape shape = new Shape(null) {...}; OR provide the no-argument constructor in the Shape class: Shape(){}

Correction Advanced Java Class Design - 6

E - Printer2

print() method is defined in interface Printer1 and class Printer2.

class Printer inherits both the methods but there is no conflict in this case as print() method defined in Printer2 class is used.

‘Printer2’ is printed on to the console.

Correction Advanced Java Class Design - 7

B -

```java
class Inner {
  public void printName() {
    System.out.println(name);
  }
}
```

name can be referred either by name or Outer.this.name. There is no keyword with the name ‘inner’ in java.

As new Inner() is used in main method, hence cannot declare class Inner as abstract in this case. But note abstract or final can be used with regular inner classes.

Keyword ‘this’ inside Inner class refers to currently executing instance of Inner class and not the Outer class.

To access Outer class variable from within inner class you can use these 2 statements:

System.out.println(name);

OR

System.out.println(Outer.this.name);

Correction Advanced Java Class Design - 8

A - None of the other options

new Flags() tries to invoke enum constructor but Enum constructor cannot be invoked.

Correction Advanced Java Class Design - 9

B - 300

Outer class (M) code has access to all the members of inner class (N) including private members, hence inner.num2 doesn’t cause any compilation error.

Correction Advanced Java Class Design - 10

D - An enum can implement interfaces

Java enums cannot extend from another class or enum but an enum can implement interfaces.

All java enums implicitly extend from java.lang.Enum class and not from java.util.Enum class.

Correction Advanced Java Class Design - 11

D - Compilation error

class Inner is method local inner class and it is accessing parameter variable x.

Starting with JDK 8, a method local inner class can access local variables and parameters of the enclosing block that are final or effectively final.

But the statement System.out.println(++x); tries to increment the value of variable x and hence compilation error.

Correction Advanced Java Class Design - 12

D - Compilation error for Printer

Starting with JDK 8, a Java interface can have default and static methods. So, no issues in Printer1 and Printer2 interfaces.

Printer class inherits both the default methods and hence compilation error.

To resolve this error, override the print() method in Printer class:

```java
public void print() {
    System.out.println("Printer");
}
```

To invoke print() method of Parent interfaces from within the overriding method, you can use below syntax:

```java
public void print() {
    System.out.println("Printer");
    Printer1.super.print();
    Printer2.super.print();
}
```

Correction Advanced Java Class Design - 13

A - Operator opr = (x, y) -> x + y;

B - Operator opr = (int x, int y) -> { return x + y; };

C - Operator opr = (x, y) -> { return x + y; };

Operator opr = (int x, int y) -> { return x + y; }; => Correct, operate(int, int) method accepts two int type parameters and returns the addition of passed parameters.

Operator opr = (x, y) -> { return x + y; }; => Correct, type is removed from left part, type inference handles it.

Operator opr = (x, y) -> return x + y; => Compilation error, if there is only one statement in the right side then semicolon inside the body, curly brackets and return statement(if available) can be removed. But all should be removed. You can’t just remove one and leave others.

Operator opr = (x, y) -> x + y; => Correct, semicolon inside the body, curly brackets and return statement, all 3 are removed from right side.

Operator opr = x, y -> x + y; => Compilation error, if there are no parameters or more than one parameter available, then round brackets cannot be removed from left side.

Correction Advanced Java Class Design - 14

A - Anonymous

System.out.println(new Object()); invokes the toString() method defined in Object class, which prints fully qualified class name, @ symbol and hexadecimal value of hash code [Similar to java.lang.Object@15db9742]

In the given code, an instance of anonymous class extending Object class is passed to System.out.println() method and the anonymous class overrides the toString() method.

Thus, at runtime overriding method is invoked, which prints "Anonymous" to the console.

Correction Advanced Java Class Design - 15

A - Hello

obj refers to an anonymous inner class instance extending from Greetings class and the anonymous inner class code correctly overrides greet() method.

Code executes and prints Hello on to the console.

Correction Advanced Java Class Design - 16

C - Compilation error

Enum constant list must be the first item in an enum.

GREEN(“go”), AMBER(“slow”), RED(“stop”); should be the first line inside TrafficLight enum.

Correction Advanced Java Class Design - 17

A - Compilation error

variable i a is local variable and it is used in the lambda expression. So, it should either be final or effectively final.

The last statement inside main(String []) method, increments value of i, which means it is not effectively final and hence compilation error.

Correction Advanced Java Class Design - 18

A - HELLO is printed twice

Enum constructor is invoked once for every constant.

For ‘Flags.TRUE’, enum constructor is invoked for TRUE as well as FALSE.

Correction Advanced Java Class Design - 19

C - INNER

invokeInner() is instance method of outer class, X. So, implicit ‘this’ reference is available for this method. this reference refers to the currently executing instance of outer class, X.

So Java compiler converts Y obj = new Y(); to Y obj = this.new Y(); and hence this syntax has no issues. So Line 9 is fine.

Because of the special relationship between Outer and inner class, Outer and Inner class can very easily access each other’s private members. Hence, no issues with Line 10 as well.

Given code compiles and executes fine and prints INNER to the console.

Correction Advanced Java Class Design - 20

B - Outer.Inner.greetings(“HELLO!”);

C -

```
Outer.Inner inner2 = new Outer.Inner();
inner2.greetings("HELLO! ");
```

Outside of top-level class, Outer, static nested class can be referred by using TOP-LEVEL-CLASS.STATIC-NESTED-CLASS. So, in this case correct way to refer static nested class is Outer.Inner.

greetings(String) is a static method, so it can be invoked by using the class name, which is by the statement: Outer.Inner.greetings(“…”);

Even though it is not preferred to invoke static method in non-static manner, but you can use the instance of class to invoke its static method.

To Create the instance of static nested class, syntax is: new TOP-LEVEL-CLASS.STATIC-NESTED-CLASS(…);

in this case, new Outer.Inner();

Correction Advanced Java Class Design - 21

A -

```
Test.A a2 = new Test().new A();
a2.m();
```

D -

```
A a1 = new Test().new A();
a1.m();
```

There are 2 parts: 1st one is referring the name of inner class, A and 2nd one is creating the instance of inner class, A.

main method is inside Test class only, so inner class can be referred by 2 ways: A or Test.A.

As, A is Regular inner class, so instance of outer class is needed for creating the instance of inner class. As keyword ‘this’ is not allowed inside main method, so instance of outer class, Test can only be obtained by new Test(). Instance of inner class can be created by: new Test().new A();

Also note, keyword ‘this’ is not allowed static main method.

Correction Advanced Java Class Design - 22

A - Yes

interface I2 is implicitly public and static (Nested interface).

Class A1 is implicitly public and static (Nested class).

Class A2 is implicitly public and static (Nested class).

You cannot explicitly specify protected, default and private for nested classes and nested interfaces inside an interface.

Correction Advanced Java Class Design - 23

C - Java

In this example, inner class's variable var shadows the outer class's variable var. So output is Java.

Few points to note here:

1. If inner class shadows the variable of outer class, then Java compiler prepends 'this.' to the variable. System.out.println(var); is replaced by System.out.println(this.var);

2. If inner class does not shadow the variable of outer class, then Java compiler prepends "outer_class.this." to the variable. So, if class Q doesn't override the variable of class P, then System.out.println(var); would be replaced by System.out.println(P.this.var);

In the given example, if you provide System.out.println(P.this.var); inside print() method, then output would be 100.

Correction Advanced Java Class Design - 24

A - Compilation error at Line 15

enums JobStatus and TestResult are siblings as both implicitly extend from java.lang.Enum class.

Siblings cannot be compared using double equals operator (==).

equals(Object) method accepts any instance which Object can refer to, which means all the instances.

Correction Advanced Java Class Design - 25

A - Compilation error

@Override annotation is used for overriding method.

But PrintMessage (P in upper case) method of anonymous inner class doesn’t override printMessage (p in lower case) method of its super class, Message.

Hence, @Override annotation causes compilation error.

Correction Advanced Java Class Design - 26

B - Printer1

Starting with JDK 8, a Java interface can have default and static methods. So, no issues in Printer1 and Printer2 interfaces.

Printer class implements both the interfaces: Printer1 and Printer 2. static method cannot be overridden, hence doesn’t conflict with default method.

No compilation error anywhere. On execution default method is invoked and ‘Printer1’ is printed on to the console.

If you want to invoke static method defined in Printer2 interface, then use: Printer2.print();

Correction Advanced Java Class Design - 27

A - @FunctionalInterface

@FunctionalInterface annotation is used to tag a functional interface.

Correction Advanced Java Class Design - 28

B -

```
Outer.Inner obj1 = new Outer().new Inner();
obj1.m();
```

There are 2 parts: 1st one is referring the name of inner class, Inner and 2nd one is creating an instance of inner class, Inner.

Now main method is outside Outer class only, so inner class can be referred by one way only and that is by using outer class name: Outer.Inner.

As, Inner is Regular inner class, so instance of outer class is needed for creating the instance of inner class. Instance of outer class, Outer can only be obtained by new Outer(). So, instance of inner class can be created by: new Outer().new Inner();

Also note, keyword ‘this’ is not allowed static main method.

Correction Advanced Java Class Design - 29

B -

```
printPrice(new Sellable() {
  @Override
  public double getPrice() {
    return 45.34;
  }
});
```

C - printPrice(null);

Instance of anonymous inner class can be assigned to static variable, instance variable, local variable, method parameter and return value.

In this question, anonymous inner class instance is assigned to method parameter.

printPrice(null); => No compilation error as asked in the question but it would throw NullPointerException at runtime.

printPrice(new Sellable()); => Cannot create an instance of Sellable type.

printPrice(new Sellable() {}); => getPrice() method is not implemented.

Correction Advanced Java Class Design - 30

C - method(s -> System.out.println(s.toUpperCase()), “good morning!”);

Lambda expression can be assigned to static variable, instance variable, local variable, method parameter or return type. method(I10, String) accepts two arguments, hence method(s -> System.out.println(s.toUpperCase())); would cause compilation error.

When curly brackets are used then semicolon is necessary, hence method(s -> { System.out.println(s.toUpperCase()) }, “good morning!”); would cause compilation error.

method(s -> s.toUpperCase(), “good morning!”); is a legal syntax but, nothing is printed to the console.

Correction Advanced Java Class Design - 31

D - None of the other options

case labels accept constant names only.

case TrafficLight.RED:, case TrafficLight.YELLOW: and case TrafficLight.GREEN: cause compilation error.

Correct case labels should be:

case RED:, case YELLOW: and case GREEN:

Correction Advanced Java Class Design - 32

B -

```
{
  System.out.println( HELLO”);
}
```

D -

```
Inner() {
  System.out.println( HELLO”);
}
```

static initialization block defined inside Outer class is invoked when static method sayHello is invoked.

Method-local inner class can be defined inside methods(static and non-static) and initialization blocks(static and non-static).

But like Regular inner class, method-local inner class cannot define anything static, except static final variables.

new Inner(); invokes the no-argument constructor of Inner class. So, System.out.println(“HELLO”) can either be provided inside no-argument constructor or instance initialization block.

Correction Advanced Java Class Design - 33

D - Compilation error

Regular inner class cannot define anything static, except static final variables.

In this case, static initialization block inside inner class Bar is not allowed.

Correction Advanced Java Class Design - 34

A - Compilation error at Line 10

Every enum extends from java.lang.Enum class and it contains following definition of clone method:

```
protected final Object clone() throws CloneNotSupportedException {
    throw new CloneNotSupportedException();
}
```

Every enum constant (RED, YELLOW, GREEN) is an instance of TrafficLight enum and as clone method is protected in Enum class so it cannot be accessed in com.training.ocp package using reference variable.

Correction Advanced Java Class Design - 35

C - Compilation error

Even though anonymous inner class allows to define methods not available in its super class but these methods cannot be invoked from outside the anonymous inner class code.

Reason is very simple, methods are invoked on super class reference variable (msg) which is of Message type.

And class Message is aware of the methods declared or defined within its boundary, printMessage() method in this case.

So using Message class reference variable, methods defined in sub class cannot be invoked. So, msg.PrintMessage(); statement causes compilation error.

Correction Advanced Java Class Design - 36

B - Lambda expression cannot be used in this case

Reference variable to which lambda expression is assigned is known as target type. Target type can be a static variable, instance variable, local variable, method parameter or return type.

Lambda expression doesn’t work without target type and target type must be a functional interface. Functional interface was added in JDK 8 and it contains one non-overriding abstract method.

As Greetings is abstract class, so lambda expression cannot be used in this case.

Correction Advanced Java Class Design - 37

B - Compilation error in Inner class code

static nested class cannot access non-static member of the Outer class using static reference.

Hence usage of variable j in Inner class causes compilation error.

Correction Advanced Java Class Design - 38

C - 3DPrinter

An interface can override the default method of parent interface and declare it as abstract as well.

ThreeDPrinter is a functional interface as it has one non-overriding abstract method. No issues with ThreeDPrinter interface.

Lambda syntax is correct and p.print() method invokes 'System.out.println("3DPrinter");' of lambda expression, hence '3DPrinter' is printed in the output.

Correction Advanced Java Class Design - 39

B - Compilation error

Enum constructors are implicitly private, even though you can provide private access modifier but it will be redundant.

Using ‘public’ or ‘protected’ for enum constructors is not allowed.

Correction Advanced Java Class Design - 40

C - new A().new B(null).m1();

new B() will cause compilation error as no-argument constructor is not defined in inner class B.

new A.B() is invalid syntax for creating the instance of Regular inner classes.

new A().new B(“hello”).m1(); is a valid syntax but it will print “hello” in the output and not “Hello”.

Correction Advanced Java Class Design - 41

B - No

Unlike other inner classes, an anonymous inner class can either extend from one class or can implement one interface.

It cannot extend and implement at the same time and it cannot implement multiple interfaces.

Correction Advanced Java Class Design - 42

A - I2 obj4 = x -> x*x;

If curly brackets are removed from lambda expression body, then return keyword should also be removed.

There should not be space between - and >.

For one parameter, parentheses or round brackets () can be removed.

Correction Advanced Java Class Design - 43

C - Compilation error at Line 9

Instance of method-local inner class can only be created within the boundary of enclosing initialization block or enclosing method.

B obj = new B(); is written outside the closing curly bracket of print(String) method and hence Line 9 causes compilation error.

Starting with JDK 8, a method local inner class can access local variables and parameters of the enclosing block that are final or effectively final so no issues with Line 5.

Correction Advanced Java Class Design - 44

B - No

Functional interface must have one and only one non-overriding abstract method.

boolean equals(Object) is declared and defined in Object class, hence it is not non-overriding abstract method.

@FunctionalInterface annotation causes compilation error.

Correction Advanced Java Class Design - 45

B - Compilation error

To refer to inner class name from outside the top level class, use the syntax: outer_class.inner_class.

In this case, correct syntax to refer B from Test class is: A.B and not B.

Correction Advanced Java Class Design - 46

B - Yes

interface can be nested inside a class. Class Outer is top-level class and interface I1 is implicitly static.

Static nested interface can use all 4 access modifiers(public, protected, default and private).

Correction Advanced Java Class Design - 47

A - A.B obj = new A.B();

In this case, you have to write code outside class A. B is a static nested class and outside class A it is referred by A.B.

Instance of class B can be created by ‘new A.B();’.

Correction Advanced Java Class Design - 48

A - Printable obj = (String msg) -> {System.out.println(msg);};

B - Printable obj = x -> System.out.printtln(x);

C - Printable obj = (msg) -> System.out.printlin(msg);

E - Printable obj = msg -> System.out.println(msg);

F - Printable obj = (msg) -> {System.out.println(msg);};

print(String) method accepts parameter of String type, so left side of lambda expression should specify one parameter, then arrow operator and right side of lambda expression should specify the body.

(String msg) -> {System.out.println(msg);}; => Correct.

(msg) -> {System.out.println(msg);}; => Correct, type of variable can be removed from left side. Java compiler handles it using type inference.

(msg) -> System.out.println(msg); => Correct, if there is only one statement in the right side then semicolon inside the body, curly brackets and return statement(if available) can be removed.

msg -> System.out.println(msg); => Correct, if there is only one parameter in left part, then round brackets can be removed.

x -> System.out.println(x); => Correct, any valid java identifier can be used in lambda expression.

y - > System.out.println(y); => Compilation error as there should not be any space between - and > of arrow operator.

Correction Advanced Java Class Design - 49

D - GO

args[0] refers to “GREEN” and args[1] refers to “AMBER”.

TrafficLight.valueOf(args[0]); -> TrafficLight.valueOf(“GREEN”);

GREEN is a valid enum constant, hence case label for GREEN is executed and “GO” is printed to the console.

Correction Advanced Java Class Design - 50

D - HELLO is printed once

Enum constructor is invoked once for every constant. There is only one constant, hence constructor is invoked only once.

For first ‘Flags.TRUE’, enum constructor is invoked but for later statements enum constructor is not invoked.

Correction Advanced Java Class Design - 51

C -

```
Replace /*INSERT 1*/ with {System.out.print(1);}
Replace /*INSERT 2*/ with {System.out.print(3);}
```

D -

```
Replace /*INSERT 1*/ with static {System.out.print(1);}
Replace /*INSERT 2*/ with {System.out.print(3);}
```

Regular inner class cannot define anything static, except static final variables.

So static {System.out.print(3);} will cause compilation error.

If a class contains, constructor, instance initialization block and static initialization block and constructor is invoked, then the execution order is:

static initialization block, instance initialization block and then constructor.

Correction Advanced Java Class Design - 52

C - Compilation error

Lambda expression’s variables x and y cannot redeclare another local variables defined in the enclosing scope.

Correction Advanced Java Class Design - 53

B - Compilation error

Keyword “this” inside method-local inner class refers to the instance of inner class.

In this case this.msg refers to msg variable defined inside Inner class but there is no msg variable inside Inner class. Hence, this.msg causes compilation error.

System.out.println(msg); would print B (msg shadows Outer class variable) and System.out.println(Outer.this.msg); would print A.

Correction Advanced Java Class Design - 54

A - Hello!

It is a valid anonymous inner class syntax. But anonymous inner class code doesn’t override printMessage() method of Message class rather it defines a new method PrintMessage (P in upper case).

Anonymous inner class allows to define methods not available in its super class, in this case PrintMessage() method.

But msg.printMessage(); statement invokes the printMessage method of super class, Message and thus “Hello!” gets printed in the output.

Correction Advanced Java Class Design - 55

B - 1000

Lambda expression is written inside Inner class, so this keyword in lambda expression refers to the instance of Inner class.

Hence, System.out.println(this.var); prints 1000.

Correction Advanced Java Class Design - 56

A - Lambda expression

Functional interface must have only one non-overriding abstract method but Functional interface can have constant variables, static methods, default methods and overriding abstract methods [equals(Object) method, toString() method etc. from Object class].

I4 is a Functional Interface.

Correction Advanced Java Class Design - 57

A - ->

Arrow operator (->) was added in JDK 8 for lambda expressions.

NOTE: there should not be any space between - and >.

Correction Advanced Java Class Design - 58

A - Program compiles and executes successfully but nothing is printed on to the console

Lambda expression is defined correctly but print() method is not invoked on obj reference.

So, no output.

Correction Advanced Java Class Design - 59

C - 11

No issues with lambda syntax: curly brackets and semicolon are available.

Variable i is declared within the body of lambda expression so don’t confuse it with local variable of main method.

i is declared and initialized to 10, i is incremented by 1 (i becomes 11) and finally value of i is printed.

Correction Advanced Java Class Design - 60

C - Compilation error

Semicolon is missing just before the statement i1.m1();

Wrong syntax of anonymous inner class.

Correction Advanced Java Class Design - 61

B - java.lang.Runnable

C - java.awt.event.ActionListener

E - java.util.Comparator

Comparator has only one non-overriding abstract method, compare. Runnable has only one non-overriding abstract method, run.

ActionListener has only one non-overriding abstract method, actionPerformed. Serializable and Cloneable are marker interfaces.

Correction Advanced Java Class Design - 62

C - None of the other options

enum Status will cause compilation error as constant name should be unique, but PASS is declared twice.

Correction Advanced Java Class Design - 63

C - Compilation error

As enum Directions contains more code after constant declarations, hence last constant declaration must be followed by a semicolon.

Correct constant declaration is:

NORTH(“N”), SOUTH(“S”), EAST(“E”), WEST(“W”);

Correction Advanced Java Class Design - 64

A - Compilation error

Regular inner class Bar cannot define any static methods.

Method m1() is static and hence compilation error.

NOTE: Regular inner class cannot define anything static, except static final variables.

Correction Advanced Java Class Design - 65

A -

```
A.B obj2 = new A.B();
obj2.log();
```

D -

```
B obj1 = new B();
obj1.log();
```

static nested class can use all 4 access modifiers (public, protected, default and private) and 2 non-access modifiers (final and abstract).

static nested class can contain all type of members, static as well as non-static. This behavior is different from other inner classes as other inner classes don’t allow to define anything static, except static final variables. This is the reason static nested class is not considered as inner class.

There are 2 parts in accessing static nested class: 1st one to access the static nested class’s name and 2nd one to instantiate the static nested class.

Within the top-level class, a static nested class can be referred by 2 ways: TOP-LEVEL-CLASS.STATIC-NESTED-CLASS and STATIC-NESTED-CLASS.

In this case, either use A.B or simply use B. Now for instantiating the static nested class, an instance of enclosing class cannot be used, which means in this case, I can’t write new A().new B(). Correct way to instance static nested class is: new A.B(); but as main method is inside class A, hence I can even write new B(); as well.

Top-level class can easily access private members of inner or static nested class, so no issues in invoking log() method from within the definition of class A.

Correction Advanced Java Class Design - 66

B - No

@FunctionalInterface annotation cannot be used here as interface I1 specifies two non-overriding abstract methods.

This code causes compilation error.

Correction Advanced Java Class Design - 67

D - Hello!

Message msg = new Message() {}; means msg doesn’t refer to an instance of Message class but to an instance of un-named sub class of Message class, which means to an instance of anonymous inner class.

In this case, anonymous inner class code doesn’t override printMessage() method of super class, Message.

So at runtime, msg.printMessage() method invokes the printMessage() method defined in super class (Message) and Hello! is printed to the console.

Correction Advanced Java Class Design - 68

B - Training_OCP

Dot (.) operator has higher precedence than concatenation operator, hence toUpperCase() is invoked on str2 and not on the result of (str1 + “_” + str2)