CTU training solutions CAREER SUCCESS STARTS AT CTU

Faculty of Information Technology									
I declare that I am familiar with, and will abide to the Examination rules of CTU [illegible] **Signature**	**SUBJECT NAME: Beginner Java** **SUBJECT CODE: J521**								
	FORMATIVE ASSESSMENT **Duration**: Jul 18 – 08 Aug **Date** 08 August 2023 **Total Marks**: 61 **Total pages**:				**Examiner**: Mr. Junior Mangayi **Moderator:** Mr. Newton				
	Student number								
	2	0	2	3	2	7	5		
	Surname: Poponi				**Initials**: M.P			/	%

Table of Contents

Application Question(s)

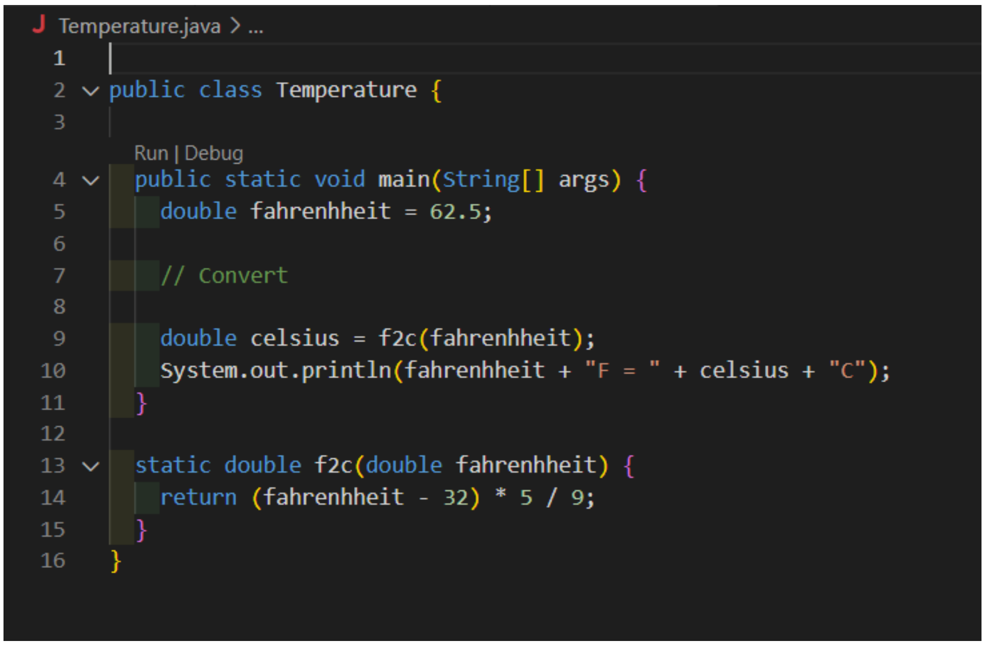
Question 1

Instructions: Code snippets below contain errors, you are required to find the error and re-write the code and fix the error. Compile it and provide the correct code snippet.

1. The following program has several errors. Modify it so that it will compile and run without errors.

```
PUBLIC CLASS temperature {
PUBLIC void main(string args) {
double fahrenheit = 62.5;
*/ Convert /*
double celsius = f2c(fahrenheit);
System.out.println(fahrenheit + 'F = ' + celsius + 'C');
}
double f2c(float fahr) {
RETURN (fahr - 32) * 5 /
```

The modified version of the program with corrections.


Temperature.java > ...

```

public class Temperature {

    public static void main(String[] args) {
        double fahrenhheit = 62.5;

        // Convert

        double celsius = f2c(fahrenhheit);
        System.out.println(fahrenhheit + "F = " + celsius + "C");
    }

    static double f2c(double fahrenhheit) {
        return (fahrenhheit - 32) * 5 / 9;
    }
}
```

Errors that were corrected on the code are as follows:

1. The class name should start with a capital letter, so I changed "temperature" to "Temperature."

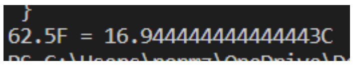
2. The `main` method signature should be written with a capital 'P' for `public`, and it should accept a `String[]` as the parameter instead of "string args."

3. In the `System.out.println` statement, I changed single quotes to double quotes for string literals.

4. The `f2c` method should take a `double` parameter, not a `float`.

5. I made the `f2c` method `static` since it's being called from the `main` method.

Output:

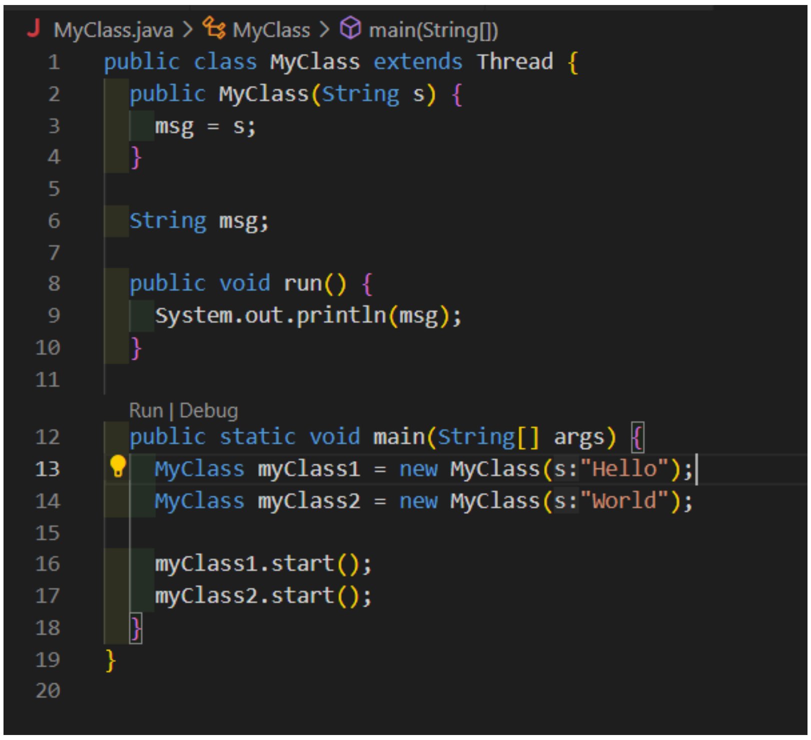


```
  }
62.5F = 16.944444444444443C
```

2. The following program has several errors. Modify it so that it will compile and run without errors

```
public class MyClass extends Thread {
public MyClass(String s) { msg = s; }
String msg;
public void run() {
System.out.println(msg);
}
public static int class (String[] args) {
New MyClass("Hello");
new MyClass("World");
}
}
```

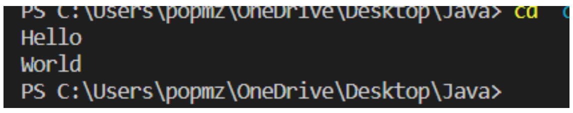
The modified version of the program with corrections.



MyClass.java > MyClass > main(String[])

```java
public class MyClass extends Thread {
    public MyClass(String s) {
        msg = s;
    }

    String msg;

    public void run() {
        System.out.println(msg);
    }

    Run | Debug
    public static void main(String[] args) {
        MyClass myClass1 = new MyClass(s:"Hello");
        MyClass myClass2 = new MyClass(s:"World");

        myClass1.start();
        myClass2.start();
    }
}
```

Output:



```
PS C:\Users\popmz\OneDrive\Desktop\Java> cd
Hello
World
PS C:\Users\popmz\OneDrive\Desktop\Java>
```

Errors that were corrected on the code are as follows:

1. Added a body to the `run` method, which is necessary for the `Thread` class.

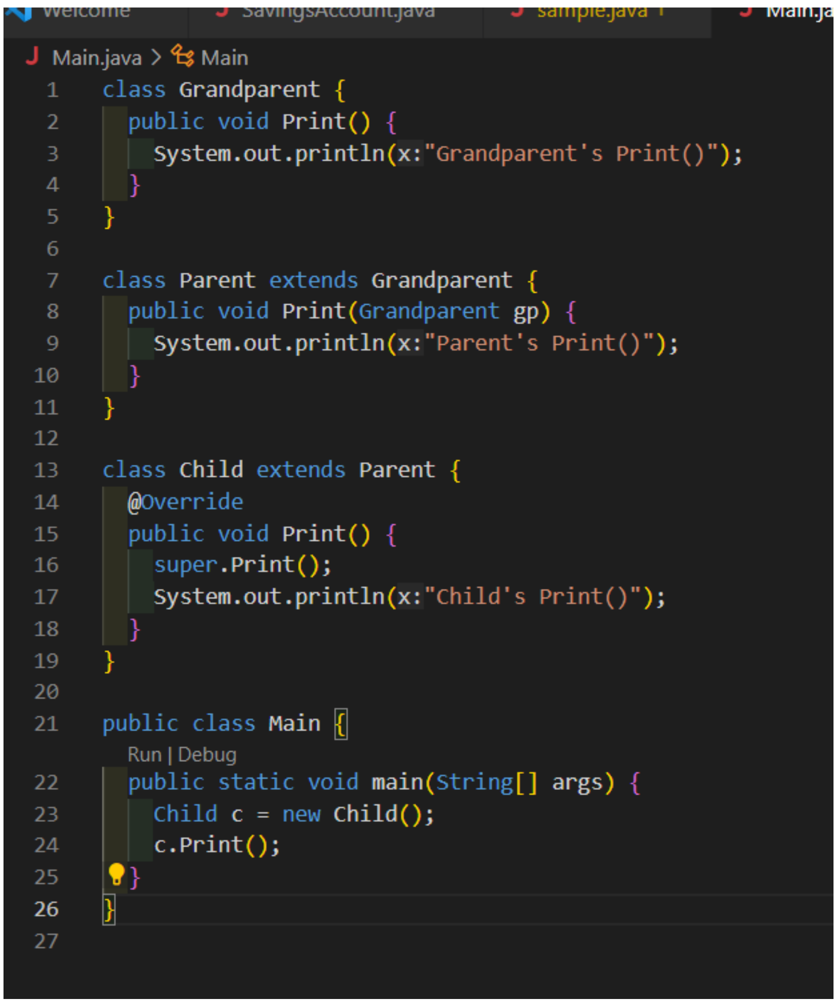
2. Renamed the `class` method to `main`.

3. Created two `MyClass` objects, `myClass1` and `myClass2`, and started them using the `start` method to execute the `run` method in separate threads.

3. The following program has several errors. Modify it so that it will compile and run without errors

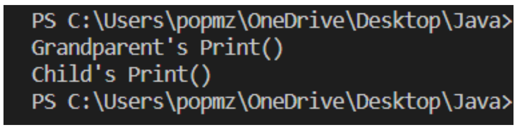
```
// filename Main.java
class Grandparent {
 public void Print() {
 System.out.println("Grandparent's Print()");
 }
}

class Parent extends main {
 public void Print(Grandparent) {
 System.out.println("Parent's Print()");
 }
}

class Child extends Parent {
 public void Print() {
 super.super.Print();
 System.out.println("Child's Print()");
 }
}

public class Main {
 public static void main(String[] args) {
 Parent c = new Child();
 c.Print();
 }
}
```

The modified version of the program with corrections.



```java
class Grandparent {
    public void Print() {
        System.out.println(x:"Grandparent's Print()");
    }
}

class Parent extends Grandparent {
    public void Print(Grandparent gp) {
        System.out.println(x:"Parent's Print()");
    }
}

class Child extends Parent {
    @Override
    public void Print() {
        super.Print();
        System.out.println(x:"Child's Print()");
    }
}

public class Main {
    public static void main(String[] args) {
        Child c = new Child();
        c.Print();
    }
}
```

Output:



```
PS C:\Users\popmz\OneDrive\Desktop\Java>
Grandparent's Print()
Child's Print()
PS C:\Users\popmz\OneDrive\Desktop\Java>
```

Identified problems in the code:

1. The `super.super.Print()` statement is invalid because you cannot use `super` to access a method in the grandparent class directly.

2. The method `Print(Grandparent)` in the `Parent` class should be written as `Print(Grandparent gp)` and should take an argument.

3. You should call the `Print` method of the `Child` class without any arguments in the `c.Print()` statement in the `main` method.

Errors that were corrected on the code are as follows:

1. `Parent` now extends `Grandparent`, allowing the `super.Print()` call to work correctly.

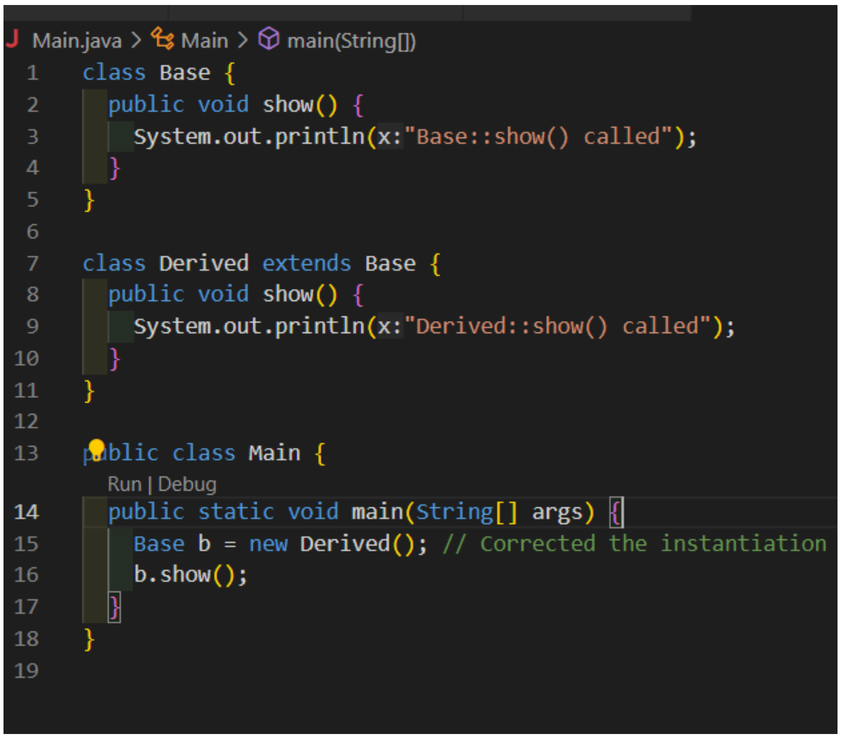
2. The `Print` method in `Parent` now takes a `Grandparent` argument (`Print(Grandparent gp)`).

3. The `Print` method in `Child` overrides the `Parent` method and calls `super.Print()` to invoke the `Print` method from the `Parent` class, and then it adds its own functionality.

4. The following program has several errors. Modify it so that it will compile and run without errors.

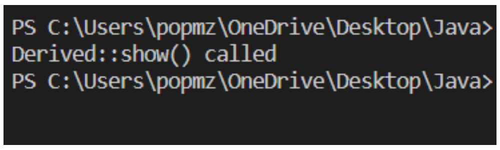
```
class Base {
 public void show() {
 System.out.println("Base::show() called");
 }
}

class Derived extends main {
 public void show() {
 System.out.println("Derived::show() called");
 }
}

public class Main {
 public static void base(String[] args) {
 Base b = new Derived();;
 b.show();
 }
}
```

The modified version of the program with corrections.



```java
class Base {
    public void show() {
        System.out.println(x:"Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
        System.out.println(x:"Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived(); // Corrected the instantiation
        b.show();
    }
}
```

Output:



```
PS C:\Users\popmz\OneDrive\Desktop\Java>
Derived::show() called
PS C:\Users\popmz\OneDrive\Desktop\Java>
```

Errors that were corrected on the code are as follows:

1. Changed `class Derived extends main` to `class Derived extends Base` to properly inherit from the `Base` class.

2. Changed `public static void base(String[] args)` to `public static void main(String[] args)` to provide the correct entry point for the Java program.

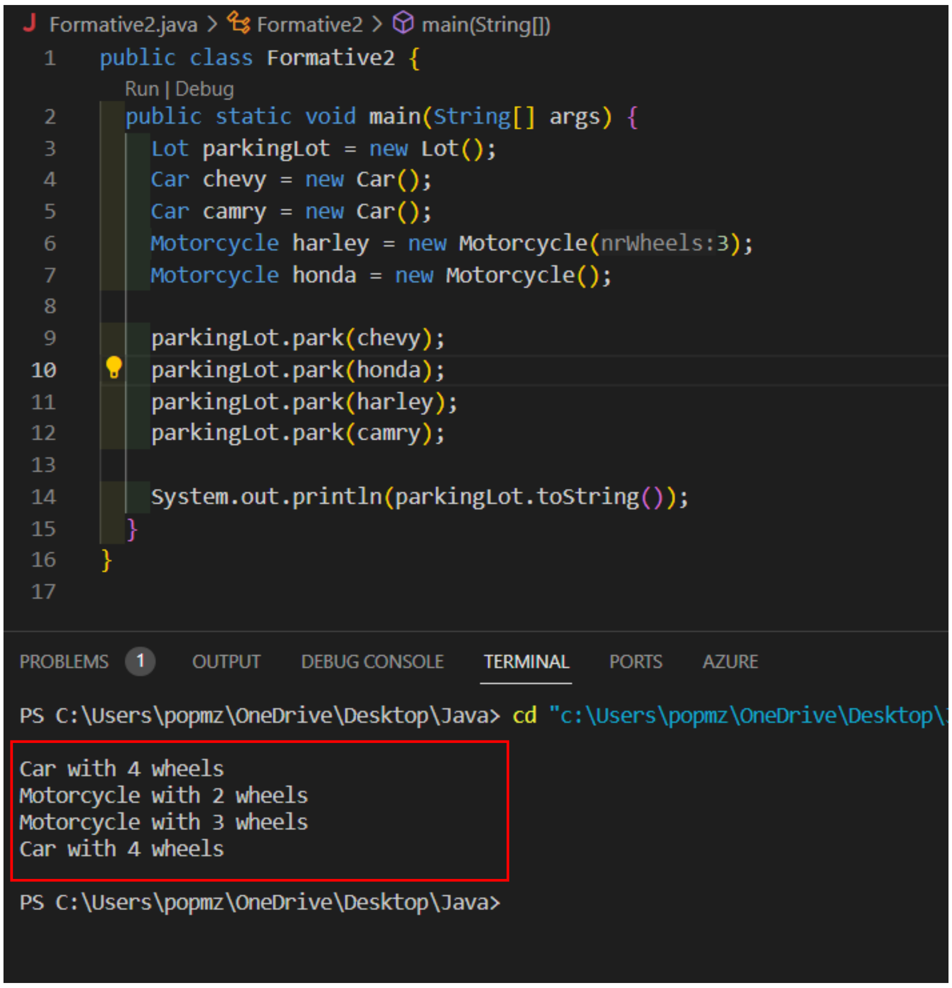
3. Removed the extra semicolon after `Base b = new Derived();` to correct the syntax.

Question 2

1. What feature(s) of this code tells you that dynamic binding is taking place?

Dynamic binding resolves method calls at runtime based on the object's type, not the variable's declared type. The Lot class's park() method and totalWheels() method utilize dynamic binding by accepting Vehicle objects as parameters, which are superclasses of Car and MotorCycle. The appropriate implementation of the toString() method for the actual object type is called at runtime, and the total number of wheels is determined based on the actual object types of the vehicles in the lot.

2. What is the output from main() ?



```java
public class Formative2 {
    public static void main(String[] args) {
        Lot parkingLot = new Lot();
        Car chevy = new Car();
        Car camry = new Car();
        Motorcycle harley = new Motorcycle(nrWheels:3);
        Motorcycle honda = new Motorcycle();

        parkingLot.park(chevy);
        parkingLot.park(honda);
        parkingLot.park(harley);
        parkingLot.park(camry);

        System.out.println(parkingLot.toString());
    }
}
```

```
PS C:\Users\popmz\OneDrive\Desktop\Java> cd "c:\Users\popmz\OneDrive\Desktop\
Car with 4 wheels
Motorcycle with 2 wheels
Motorcycle with 3 wheels
Car with 4 wheels

PS C:\Users\popmz\OneDrive\Desktop\Java>
```

3. True/False -- The Vehicle class is an abstract class.
   False
4. True/False -- Motorcycle is derived from Vehicle.
   True
5. True/False -- Car is derived from Vehicle.
   True
6. True/False -- setWheels() cannot be used polymorphically.
   False
7. When a Car object is created, which constructors are invoked, and in what order?

   When a Car object is created, the following constructors are invoked in this order:

   1. The Vehicle class constructor with an int parameter: The Car class inherits from the Vehicle class and calls its constructor with the super() keyword. Since the Vehicle class has a constructor that takes an int parameter, that constructor is invoked first. This sets the number of wheels for the Car object to 4.

   2. The Car class constructor with no parameters: After the Vehicle constructor completes execution, the Car class constructor with no parameters is invoked. This initializes any properties or behaviour specific to the Car class.

      Therefore, the output of Car's toString() method would be "Car with 4 wheels"

8. What is the purpose of the instance variable nrVehicles in the Lot class?

   The instance variable nrVehicles in the Lot class is used to keep track of the number of vehicles currently parked in the parking lot. It is initialized to zero in the constructor, and its value is incremented each time a vehicle is parked using the park() method.

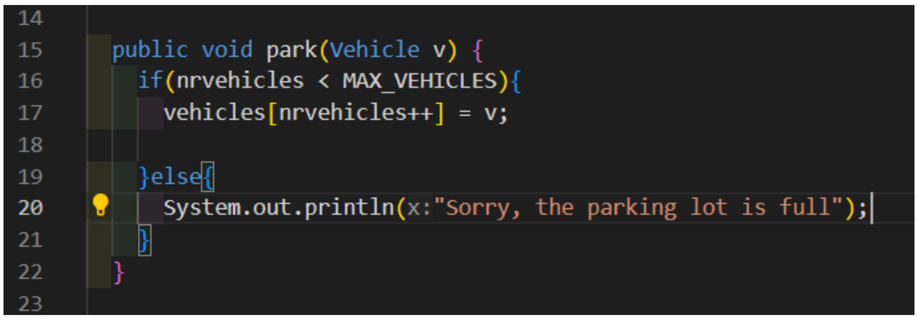
   The nrParked() method returns the current number of parked vehicles by simply returning the value of nrVehicles.

   The totalWheels() method uses the nrVehicles variable to iterate through the parked vehicles and accumulate the total number of wheels in the parking lot.

The toString() method also uses nrVehicles to iterate through the parked vehicles and generate a string representation of each vehicle using the toString() method of the Vehicle, Car, and MotorCycle classes. The string representation of each vehicle is then concatenated with the newline character (\n) to produce the final string that represents the entire parking lot.

9. Identify the common coding mistake in Lot's park method. How would you correct this coding error?

Corrected code:



```
public void park(Vehicle v) {
  if(nrvehicles < MAX_VEHICLES){
    vehicles[nrvehicles++] = v;

  }else{
    System.out.println(x:"Sorry, the parking lot is full");
  }
}
```

10. In the Vehicle constructor, what is the purpose of { this(4); } ?

In the Vehicle constructor, the statement {this(4)} calls another constructor within the same class, passing the value 4 as an argument. This is called a constructor chaining. The purpose of this statement is to initialize the (nrWheels) instance variable with a default value of 4 if no argument is provided when creating a new Vehicle object. This avoids duplicating code by allowing the two constructors in the Vehicle class to share common initialization code.

11. In the MotorCycle constructor, what is the purpose of { super(nrWheels); }?

In the `MotorCycle` constructor, the line `super(nrWheels)` is calling the constructor of the parent class `Vehicle`, passing the `nrWheels` parameter. This is done using the `super` keyword, which is used to refer to the parent class. The purpose of this line is to set the number of wheels of the motorcycle object being created, based on the `nrWheels` parameter passed to the `MotorCycle` constructor. This allows the `MotorCycle` object to inherit the `nrWheels` field from the `Vehicle` class, which it extends, and to set its value based on the number of wheels specified in the constructor.

12. Car's toString() method invokes getWheels(), even though getWheels( ) is not defined in the Car class. How is this possible?

This is possible because the Car class inherits the getWheels() method from its superclass, Vehicle. Since Car is a subclass of Vehicle, it has access to all of the public and protected methods and variables of the Vehicle class, including the getWheels() method. Therefore, when the toString() method of the Car class invokes getWheels(), it is actually calling the getWheels() method defined in the Vehicle class.