Georgia Sugisandhea_535230080_Kelas C

1. What is the difference between a class declaration and an object declaration (the latter also known as an instantiation)?

    Class is basically like a blueprint/recipe to create objects. So, objects created from this class share common properties and methods. A class declaration is declaring the class (the recipe), including the name of the class, attributes like superclass, and whether it is public/private, final, or abstract. While an object declaration is declaring the variable of the object created from the recipe(class) with a new variable name and an object type (the class). So a class declaration is declaring the recipes to make the objects with all the details, and object declaration is declaring the creation from the recipes with the object type and the name.

2. What is an instance variable?

    Instance variables is a term used to mean an internal variable maintained by an instance. Each instance has its own collection instance variables. It is a variable where their values should not be changed directly by clients but by methods associated with the class. Shortly, it is a variable in and maintained by an instance.

3. What are the two most basic aspects of a class?

    Methods and Data Fields

4. What is the meaning of the modifiers final and static in Java? How do these two features combine to form a symbolic constant?

    Modifiers final means that the data field is constant or immutable. Which means that once set, that final value of the data field isn't able to be subsequently be changed. While static is used for a common data field that is shared by all instances of a class. So, for the entire class, there is only one copy the static variable. These two features combine into a symbolic constant, whereas symbolic constant means the value of this variable is the same for all instances of the class and can't be changed after assigning them, or in short constant values. A combination of both final (immutable value) and static (value used by all instances).

5. What does the term public mean? What does the term private mean?

    Public and private are visibility modifiers. Public means features that are known and can be used outside of the class definition. While private is featured that can only be used within a class definition, which guarantees that the only way the data fields will be modified is by methods associated with the class.

6. What is a constructor?

    Constructor is a special method that has the same name as the class and is used to initialize the data fields in an object. A constructor is called when creating an object of a class.

7. What is an accessor method? What is the advantage of using accessor methods instead of providing direct access to a data field?

   Accessor is a method that provides access to a data field that is labeled as private. It is a method that does nothing more than return the value of a data field. Sometimes termed as a getter. The advantage is that accessor method makes the data field read-only, while a data field is actually can be both read and written. So, a combination of private data field and an accessor is ensuring that the value cannot be changed unless through the accessor.

8. What is a mutator, or setter, method?

   Mutator or setter method is a method whose major purpose is simply to set a value. Usually, a setter generally named beginning with the word set.

9. What are some guidelines for selecting the order of features in a class definition?
   - Important features should be listed earlier in the class definition, less important features listed later.
   - Constructors are one of the most important aspects of an object definition and hence should appear very near the top of a class definition.
   - The declaration of methods should be grouped to facilitate finding the
   - body associated with a given message selector. Ways of doing this include listing methods in alphabetical order or grouping methods by their purpose.
   - Private data fields are important only to the class developer. They should be listed near the end of a class definition.

10. What is an immutable data field?

    An immutable data field means that once that data field is set, the value of the data field is not able to be subsequently be changed.

11. What is a fully qualified name?

    A fully-qualified class name is a name where it contains the package that the class originated from. Fully-qualified class name is able to be obtained using getName() method.

12. How is an interface different from a class? How is it similar?

    An interface is similar to a class, for both class and interface defines the protocol for certain behavior. Both class and interface define a new type. But while both of them defines a new type, class defines instance methods with full implementation, while interfaces could only define a certain behavior without providing the implementation.

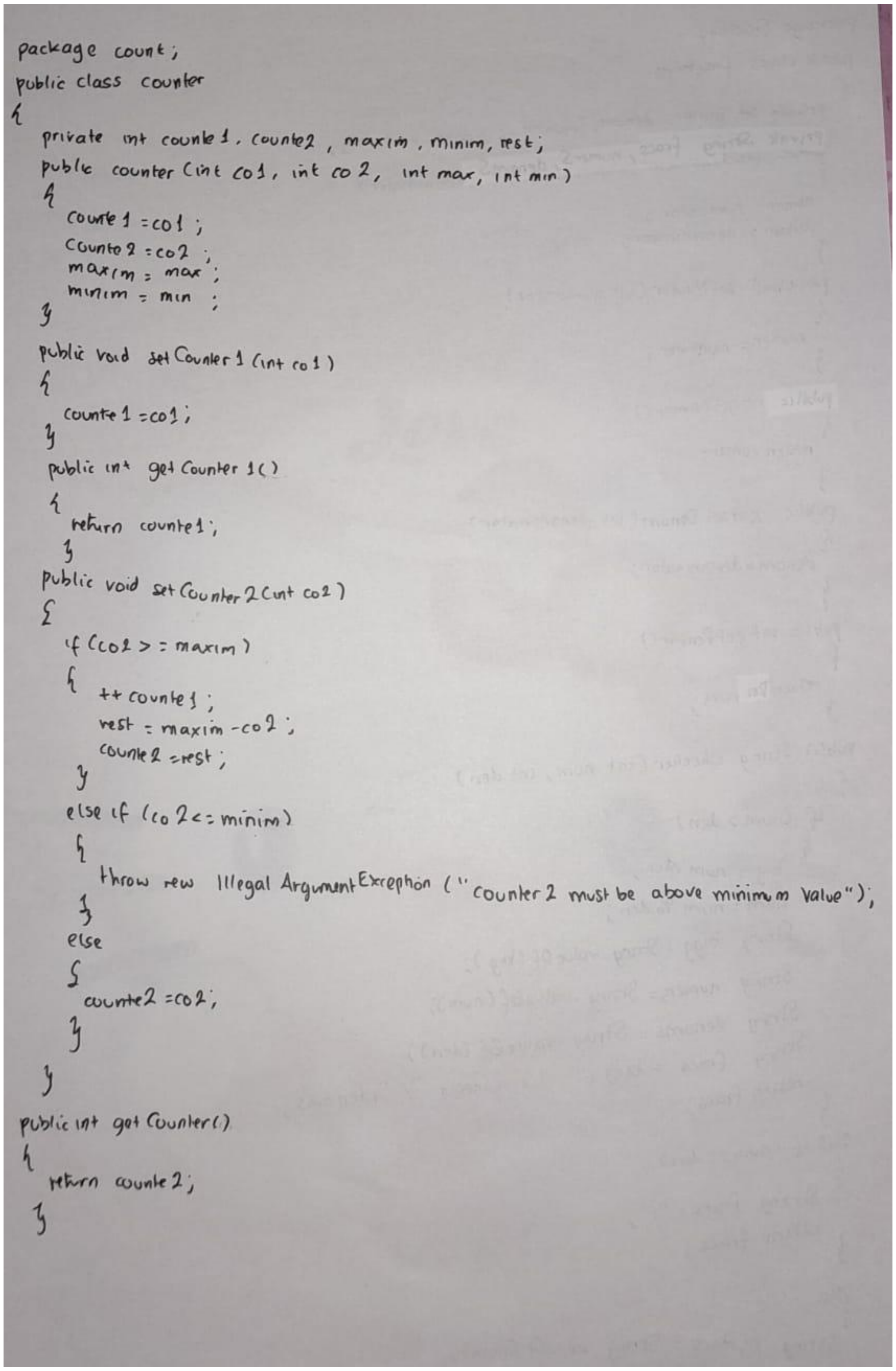
13. What is an inner or nested class?

    Inner or nested class is a class definition inside of another. In Java, inner class is linked to a specific instance of the surrounding class and is permitted access to data fields and methods in this object.

14. Explain the paradox arising from the initialization of common data fields or class data fields.

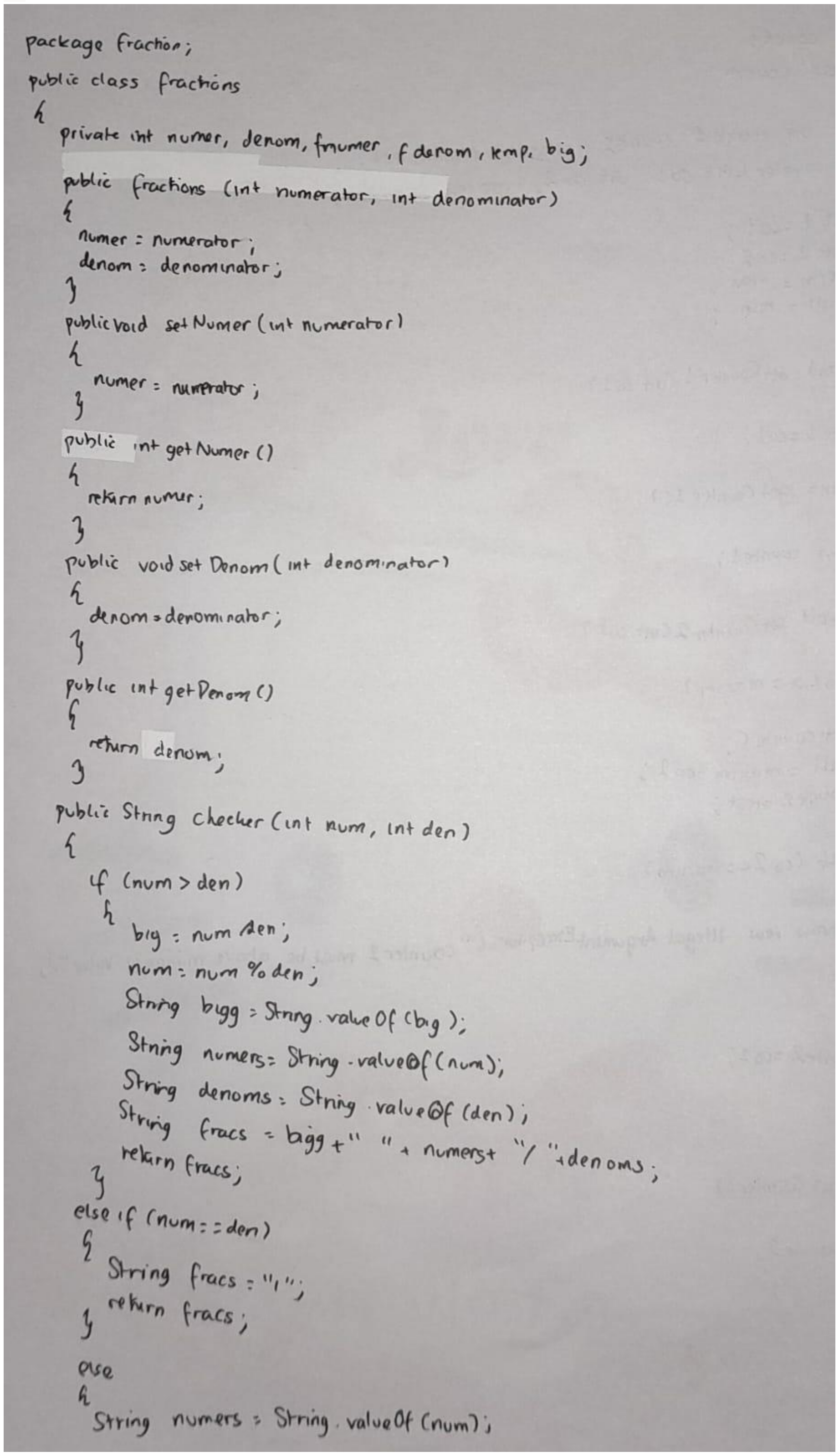
Classes are made to reduce the amount of work necessary when it comes to create similar objects. Say that we defined a common data area that is shared by all instances of a class, and think about the task of initializing this common area. But a paradox occurred when this initialization of one static object depends on the initialization of another static object, but the order of the initialization isn't guaranteed. Because this can lead to undefined behavior and unexpected results, to resolve this paradox, it requires moving outside the class/method/instance. It's resolved into another mechanism, not the object themselves. If the objects are automatically initialized to a special value by the memory manager, then every instance can test for this special value and perform the initialization regardless which one is first. Or there is a better technique such as static modifier, symbolic constants, which is executed when the class is loaded.

Exercise
3.



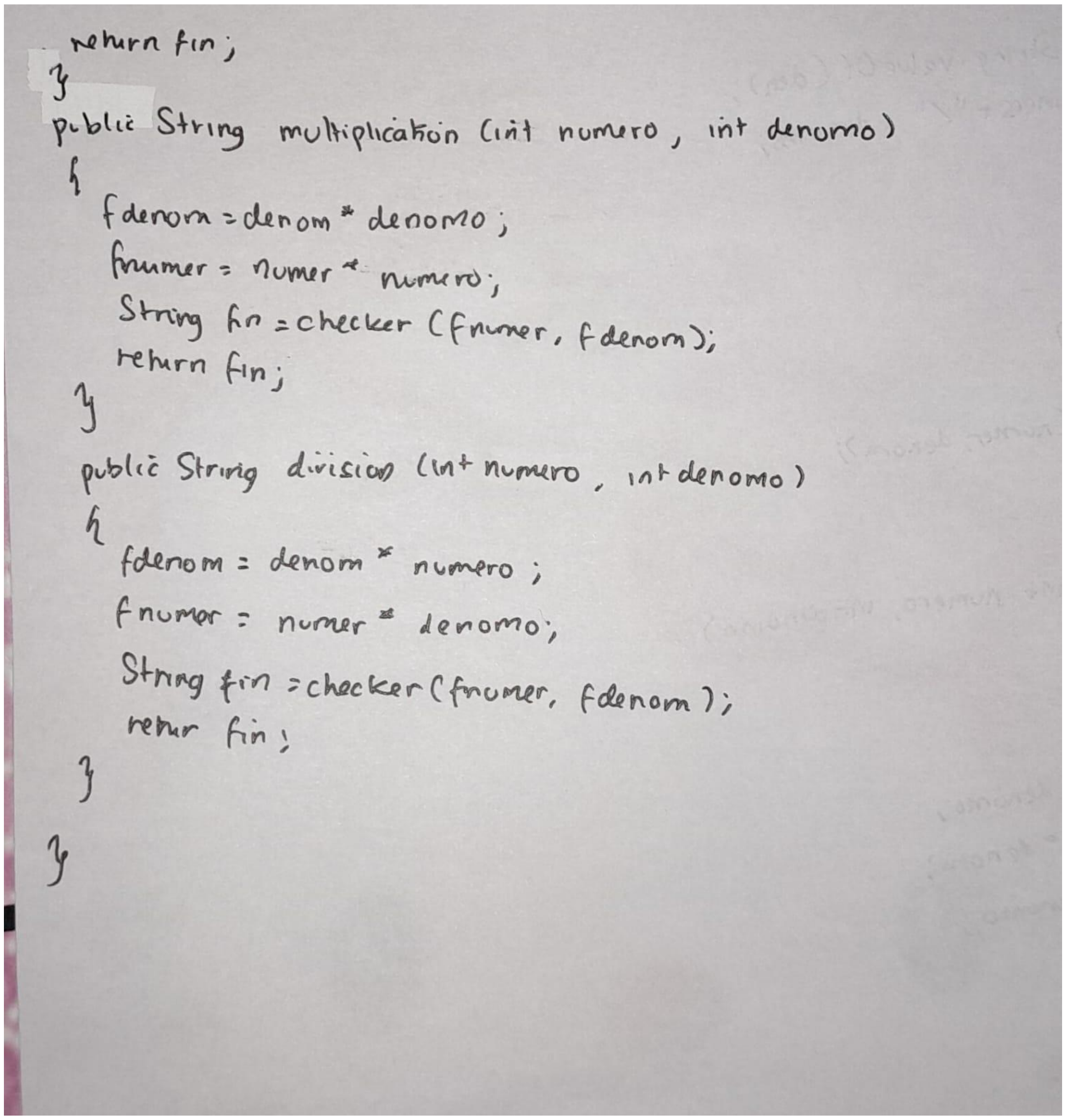
```
package count;
public class counter
{
  private int counte1, counte2 , maxim , minim, rest;
  public counter (int co1, int co2, int max, int min)
  {
    counte1 =co1 ;
    counte2 =co2 ;
    maxim = max ;
    minim = min ;
  }
  public void setCounter1 (int co1)
  {
    counte1 =co1;
  }
  public int getCounter1()
  {
    return counre1;
  }
  public void setCounter2(int co2)
  {
    if (co2 >= maxim)
    {
      ++ counte1 ;
      rest = maxim -co2 ;
      counte2 =rest ;
    }
    else if (co2<=minim)
    {
      throw new IllegalArgumentException ("counter 2 must be above minimum value");
    }
    else
    {
      counte2 =co2;
    }
  }
public int getCounter()
{
  return counte2;
}
```

5.



```
package fraction;
public class fractions
{
  private int numer, denom, fnumer, fdenom, temp, big;
  public fractions (int numerator, int denominator)
  {
   numer = numerator;
   denom = denominator;
  }
  public void setNumer (int numerator)
  {
    numer = numerator;
  }
  public int getNumer ()
  {
    return numer;
  }
  public void setDenom ( int denominator)
  {
    denom = denominator;
  }
  public int getDenom ()
  {
    return denom;
  }
 public String checker (int num, int den)
  {
    if (num > den)
    {
      big = num /den;
      num = num % den;
      String bigg = String.valueOf (big);
      String numers = String.valueOf(num);
      String denoms = String.valueOf (den);
      String fracs = bigg + " " + numers + "/ " + denoms;
      return fracs;
    }
   else if (num == den)
   {
     String fracs = "1";
     return fracs;
   }
   else
   {
    String numers = String.valueOf (num);
```

```
        String denoms = String.valueOf(den);
        String fracs : numers + "/" + denomss;
        return fracs;
        }
    }
public String getFracs()
 {
    String fin = checker(numer, denom);
    return fin;
 }
public String addition (int numero, int denomo)
 {
    if (denomo != denom)
    {
       f denom = denom * denomo;
       f numer : numero * denom;
       temp = numer * denomo;
    }
    else
    {
       fnumer = numero;
        temp = numer;
       fdenom = denomo;
    }
    fnumer = temp + fnumer;
    String fin = checker(fnumer, fdenom);
    return fin;
 }
public String substraction (int numero, int denomo)
{
   if (denomo != denom)
    {
      fdenom = denom * denomo;
      fnumer : numero * denom;
       temp = numer * denumo;
    }
  else
  {
    fnumer = numero;
    temp = numer;
    fdenom = denomo;
  }
  fnumer = temp - fnumer;
  String fin = checker(fnumer, fdenom);
```



```
 return fin;
}
public String multiplication (int numero , int denomo)
 {
   fdenom = denom * denomo ;
   fnumer = numer * numero;
   String fin = checker (fnumer, fdenom);
   return fin;
 }
 public String division (int numero , int denomo)
 {
   fdenom = denom * numero ;
   fnumer = numer * denomo;
   String fin = checker(fnumer, fdenom );
   retur fin;
 }

}
```