CSE 1030 Introduction to Computer Science II
A Sample of Test 1

1

(a) In the declaration of an attribute, what does the keyword `final` denote?

The attribute is a constant, that is, its value can only be set once.

(b) In the declaration of an attribute, what does the keyword `static` denote?

The attribute is associated with the class, not with each instance. Hence, there is only one such attribute.

(c) In the body of a method, what does the keyword `this` denote?

It refers to the object on which the method is invoked. It is the implicit parameter of the method.

(d) In the header of a method, what does the keyword `void` denote?

The method does not return a result.

2

Give three reasons why non-static attributes should be declared as `private`.

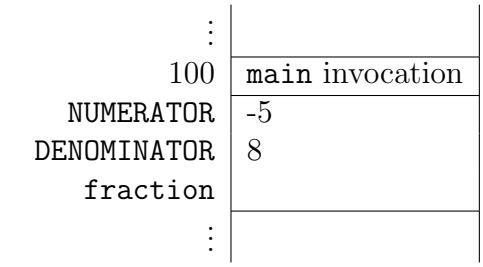
It ensures that others cannot directly access the attributes and, hence, cannot assign them illegal values (for example, setting the age of a person to a negative number). It simplifies the API (private attributes are not part of the API). It allows the implementer to change the representation of the data (without having to change the API).

3

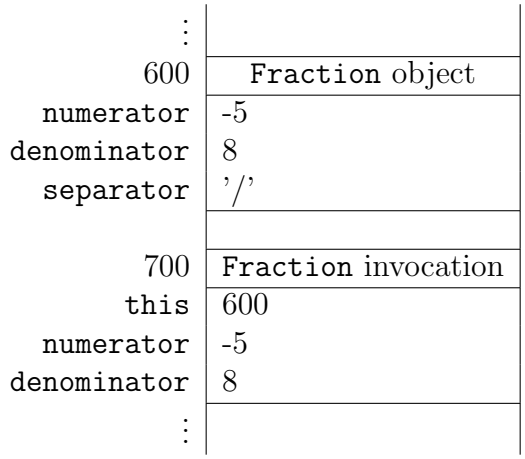
Consider the following snippet of client code.

```
1 final int NUMERATOR = -5;
2 final int DENOMINATOR = 8;
3 Fraction fraction;
4 fraction = new Fraction(NUMERATOR, DENOMINATOR);
```

Assume that once the execution reaches the end of line 3, memory can be depicted as follows.



Draw the invocation block (and any related blocks) of the constructor corresponding the invocation of the constructor of the `Fraction` class in line 4. Only draw those parts that are new or changed.



4

Consider the `Fraction` class, (a part of) the code of which can be found at the end of this test. Two fractions are considered equal if they have the same numerator and denominator. Consider the following skeleton for the `equals` method.

```
1 public boolean equals(Object object)
2 {
3    boolean equal;
4    if (object != null && ...)
5    {
6       Fraction other = (Fraction) object;
7       equal = ...;
8    }
9    else
```

```
10      {
11          equal = false;
12      }
13      return equal;
14  }
```

(a) Why is the parameter of type `Object` (and not of type `Fraction`)?

This way, the method overrides the toString method of the Object class. If the type of the parameter were Fraction, then the class would have two equals methods.

(b) Critique the method `hashCode`.

```
1  public int hashCode()
2  {
3      return (int) this.separator;
4  }
```

Discuss whether it can be improved. If so, why and how?

It does not satisfying the following property: if `x.equals(y)` *returns true then* `x.hashCode()` *and* `y.hashCode()` *return the same integer. The following method does satisfy the property.*

```
1  public int hashCode()
2  {
3      return this.numerator + this.denominator;
4  }
```

5

Consider the `Fraction` class, (a part of) the code of which can be found at the end of this test.

(a) Is `this.denominator > 0` `null` a class invariant? **Explain** your answer.

Yes. The constructor ensures that the invariant holds since it has it as a precondition. The only method that modifies the denominator attribute also has it as a precondition.

(b) Is `this.numerator > 0` a class invariant? **Explain** your answer.

No. Neither the constructor ensures it, nor the setNumerator method does not maintain it.

```
1  /**
2   * This class represents a fraction.
3   *
4   */
5  public class Fraction
6  {
7      private int numerator;
8      private int denominator;
9      private char separator;
10
11     /**
12      * Initializes a fraction with the given numerator and denominator.
13      *
14      * @param numerator the numerator of this fraction.
15      * @param denominator the denominator of this fraction.
16      * @pre. denominator > 0
17      */
18     public Fraction(int numerator, int denominator)
19     {
20         this.numerator = numerator;
21         this.denominator = denominator;
22         this.separator = '/';
23     }
24
25     /**
26      * Returns the numerator of this fraction.
27      *
28      * @return the numerator of this fraction.
29      */
30     public int getNumerator()
31     {
32         return this.numerator;
33     }
34
35     /**
36      * Sets the numerator of this fraction to the given numerator.
37      *
38      * @param numerator the new numerator of this fraction.
39      */
```

```
40      public void setNumerator(int numerator)
41      {
42          this.numerator = numerator;
43      }
44
45      /**
46       * Returns the denominator of this fraction.
47       *
48       * @return the denominator of this fraction.
49       */
50      public int getDenominator()
51      {
52          return this.denominator;
53      }
54
55      /**
56       * Sets the denominator of this fraction to the given denominator.
57       *
58       * @param denominator the new denominator of this fraction.
59       * @pre. denominator > 0
60       */
61      public void setDenominator(int denominator)
62      {
63          this.denominator = denominator;
64      }
65
66      /**
67       * Sets the separator of this fraction to the given separator.
68       *
69       * @param separator the new separator of this fraction.
70       */
71      public void setSeparator(char separator)
72      {
73          this.separator = separator;
74      }
75
76      public boolean equals(Object object) { ... }
77      public int hashCode() { ... }
78      public String toString() { ... }
79  }
```