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

1 (20 marks)

Give one reason **why** is it best to initialize *static* attributes as you declare them?

If you delay or forget their initialization, the compiler will not issue a compile-time error; it will simply assign default values to them. Relying on such defaults is not a good idea.

2 (20 marks)

Consider the `Circle` class whose API can be found at the end of this test. Three different implementers (1, 2, and 3) declare the attributes as follows.

1. ```
   private double radius;
   private double diameter;
   ```

2. ```
   private double radius;
   ```

3. ```
   double diameter;
   ```

Which choice do you like best, 1, 2, or 3? You have to **explain** your answer to receive any marks.

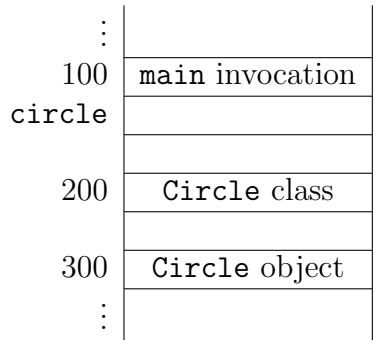
I like 2 best. I do not like 1 because either radius or diameter is redundant (given the diameter I can get the radius by dividing the diameter by two, and given the radius I can get the diameter by multiplying the radius by two). I do not like 3 since non-final attributes should be declared private.

3 (20 marks)

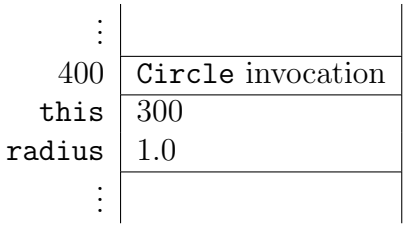
Consider the `Circle` class whose API can be found at the end of this test. Consider the following `main` method.

```
Circle circle = new Circle(1.0);
```

Just before the constructor is invoked, memory can be depicted as follows.
```



Draw the invocation block (and any related blocks) for the constructor invocation. *Only* draw those parts that are new or changed.



4 (20 marks)

Consider the `Circle` class whose API can be found at the end of this test. Consider the following `equals` method of the `Circle` class.

```
public boolean equals(Object object)
{
  if (this.getClass() == object.getClass())
  {
    Circle other = (Circle) object;
    return this.getRadius() == other.getRadius();
  }
  else
  {
    return false;
  }
}
```

Mention *three* aspects that can be improved and describe **how** they can be improved.

- *Add* `object != null` *to avoid a* `NullPointerException`.

- *Only have a single* `return` *statement.*
- *Comparison of the radii is different from the specification in the API.*

```
public boolean equals(Object object)
{
  boolean equal;
  if (object != null && this.getClass() == object.getClass())
  {
    Circle other = (Circle) object;
    final double EPSILON = 0.00001;
    equal = Math.abs(this.getRadius() - other.getRadius()) < EPSILON;
  }
  else
  {
    equal = false;
  }
  return equal;
}
```

5 (20 marks)

(a) What is the *one instance per state* design pattern?

There is only one instance of the class for every possible combination attribute values.

(b) Describe **how** the implementation of the `Circle` class needs to be modified so that the `Circle` class has at most one instance per state.

- *Make the constructor private.*
- *Introduce a static attribute that keeps track of the created* `Circle`*s.*

```
private static Map<Double, Circle> instances = new HashMap<Double, Circle>();
```

- *Introduce a static method* `getInstance` *which returns a* `Circle`*.*

```
public static Circle getInstance(double radius)
{
  Circle instance = Circle.instances.get(radius);
```

```
    if (instance == null)
    {
      instance = new Circle(radius);
      Circle.instances.put(radius, instance);
    }
    return instance;
  }
```