**Questions 1-8: MCQ**

**1.**

1. 1
2. 1.5
3. 3
4. 20
5. 20.5

**2.**

1. 1
2. 2
3. 4
4. 6
5. 10

**3.**

1. Thing someThing = new Thing("Green");
2. Thing someThing = new Thing("");
3. Thing someThing = new Thing();
4. Thing someThing;
5. Thing("Green");

**4.**

1. Fdb
2. FGdebc
3. Gec
4. GHefcd
5. There is no output due to a compilation error.

**5.**

1. a = 0,  b = 4,   and  c = 0
2. a = 0,  b = 4,   and  c = 1
3. a = 1,  b = 0,   and  c = -1
4. a = 1,  b = 1,   and  c = 0
5. a = 1,  b = 1,   and  c = 1

**6.**

1. x < 100
2. x <= 100
3. x > 10
4. x >= 10
5. x != 10

**7.**

1. Student max = new Student ("Max", 10, 3.75);
2. Student max = new Student (3.75, "Max", 10);
3. Student max = new Student (3.75, 10, "Max");
4. Student max = new Student (10, "Max", 3.75);
5. Student max = new Student (10, 3.75, "Max");

**8.**

1. In the for loop header, the initial value of j should be 0.
2. In the for loop header, the initial value of j should be 2.
3. The for loop condition should be  j < fibs.length - 1.
4. The for loop condition should be  j < fibs.length + 1.
5. The for loop should increment j by 2 instead of by 1.

**Question 9: TargetSimulation**

|  |  |  |
| --- | --- | --- |
| **Part (a)** | simulate | **5 points** |

**Part A**

**+1** Calls throwAccuracy() within the context of a loop.

**+1** Initializes and accumulates the number of balls thrown and compares the count to maxBalls.

**+1** Determine if a ball has hit the target.

**+1** Return false if maxBalls has been exceeded.

**+1** Return true if a ball has hit the target.

|  |  |  |
| --- | --- | --- |
| **Part (b)** | runSimulations | **4 points** |

**+1** Calls simulate() the specified number of times (no bounds errors).

**+1** Initializes and accumulates a count of true results.

**+1** Calculates proportion of successful simulations using double arithmetic.

**+1** Returns the calculated value.

**Question 9: TargetSimulation**

***Part (a)***

public boolean simulate() {

for (int ball = 0; ball < maxBalls; ball++) {

if (throwAccuracy() < targetRadius) return true;

}

return false;

}

***Part (b)***

public double runSimulations(int num) {

int sims = 0;

for (int n = 0; n < num; n++) {

if (simulate()) sims++;

}

return (double)sims / num;

}

**Question 10: *SleepTracker***

|  |  |  |
| --- | --- | --- |
| **Class:** | SleepTracker | **9 points** |

**+1** Declares all appropriate private instance variables

**+2** constructor

**+1** Declares header: public SleepTracker (int \_\_\_ )

**+1** Uses parameter and appropriate values to initialize instance variables

**+3** addHoursSlept method

**+1** Declares header: public void addHoursSlept( int \_\_\_\_ )

**+1** Identifies nights with hours above and nights with hours below, and increments each count

**+1** Increments the total nights count.

**+1** totalNights method

**+1** Declares and implements public int totalNights()

**+1** under method

**+1** Declares and implements public double under()

**+1** over method

**+1** Declares and implements public double over()

**Question 10: SleepTracker**

public class SleepTracker {

private int healthyMin;

private int healthyMax;

private int totalNights;

private int nightsUnder;

private int nightsOver;

public SleepTracker ( int min, int max ) {

healthyMin = min;

healthyMax = max;

totalNights = 0;

nightsUnder = 0;

nightsOver = 0;

}

public void addHoursSlept( double sleep ) {

totalNights++;

if (sleep > healthyMax) nightsOver++;

if (sleep < healthyMin) nightsUnder++;

}

public int totalNights () {

return totalNights;

}

public double under () {

if (totalNights == 0) return 0.0;

return (double)nightsUnder / (double)totalNights;

}

public double over () {

if (totalNights == 0) return 0.0;

return (double)nightsOver / (double)totalNights;

}

}