THIS IS WAR

An abstraction of a Tank exists in the form of a class.

```
public class Tank{

  //first declare the class properties

  private String id;

  private Missile[] missiles;

  int nextAvailableMissle;

  //one constructor method

  public Tank(){

    this.id = null;

    this.missiles = new Missile[10];

    this.nextAvailableMissle = -1;

  }

  //another constructor method, same name, different parameter list!

  public Tank(String id){

    this.id = id;

    this.missiles = new Missile[10];

    this.nextAvailableMissle = -1;

  }

  public void addMissile(Missile m){

    if(nextAvailableMIssile < 9){

      nextAvailableMissile = nextAvailableMissile+1;

      missiles[nextAvailableMissile] = m;

    }

  }

  public void shootMissile(){

    //code missing

  }

}
```

Explain why the Tank class is considered to be an abstraction of a tank [2]

An abstraction of a missile also exists in the form of a class.

```
public class Missile{
   //first declare the class properties
   private int damageFactor;
   //one constructor method
   public Missile(){
      this.damageFactor = (int)(Math.random()*10)+1; //between 1 and 10
   }
   //another constructor method, same name, different parameter list - method
   overloading!
   public Missile(int damageFactor){
      this.damageFactor = damageFactor;
   }
   public int getDamageFactor(){
      return this.damageFactor;
   }
   }
}
```

The `Missile` class exhibits a form of polymorphism. Name this form of polymorphism and describe it:

[3]

Draw the UML diagram describing the relationship between the `Tank` class and the `Missile` class. Make the `Missile` class UML complete but only include the title of the `Tank` class.

[3]

In a Main program, write a line of code to instantiate a new `Tank` object with the identifier `exterminator` with an id of **ext-1**. [3]

Add a `Missile` object to exterminator's `Missile` array. [2]

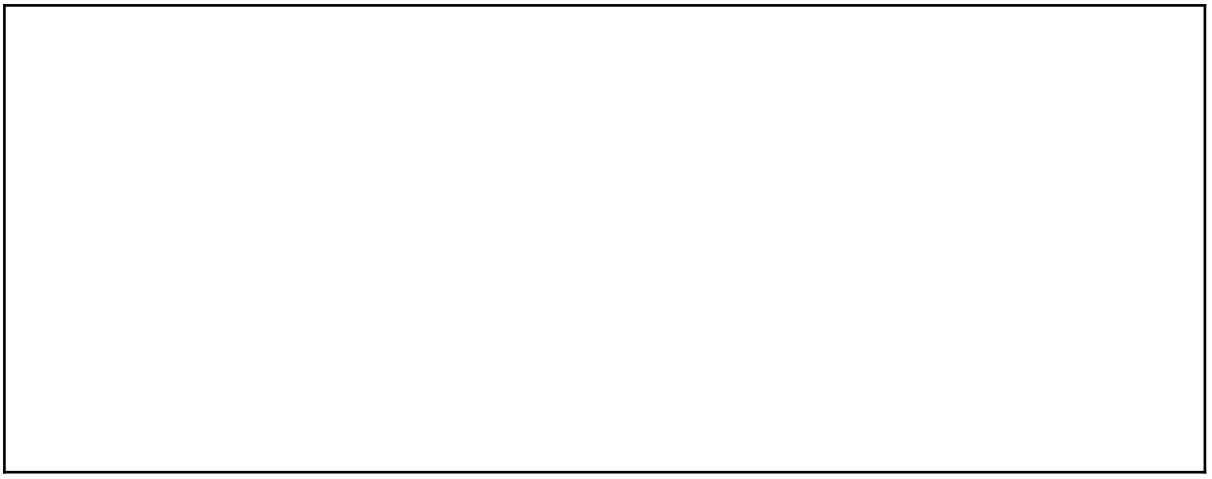
When the Tank class's `shootMissile` method is called, it:

checks that there is a missile available ie `nextAvailableMissile != -1`
if so it

- Assigns the next available missile to a temporary variable
- decrements `nextAvailableMissile`
- returns the missile reference

else it returns null

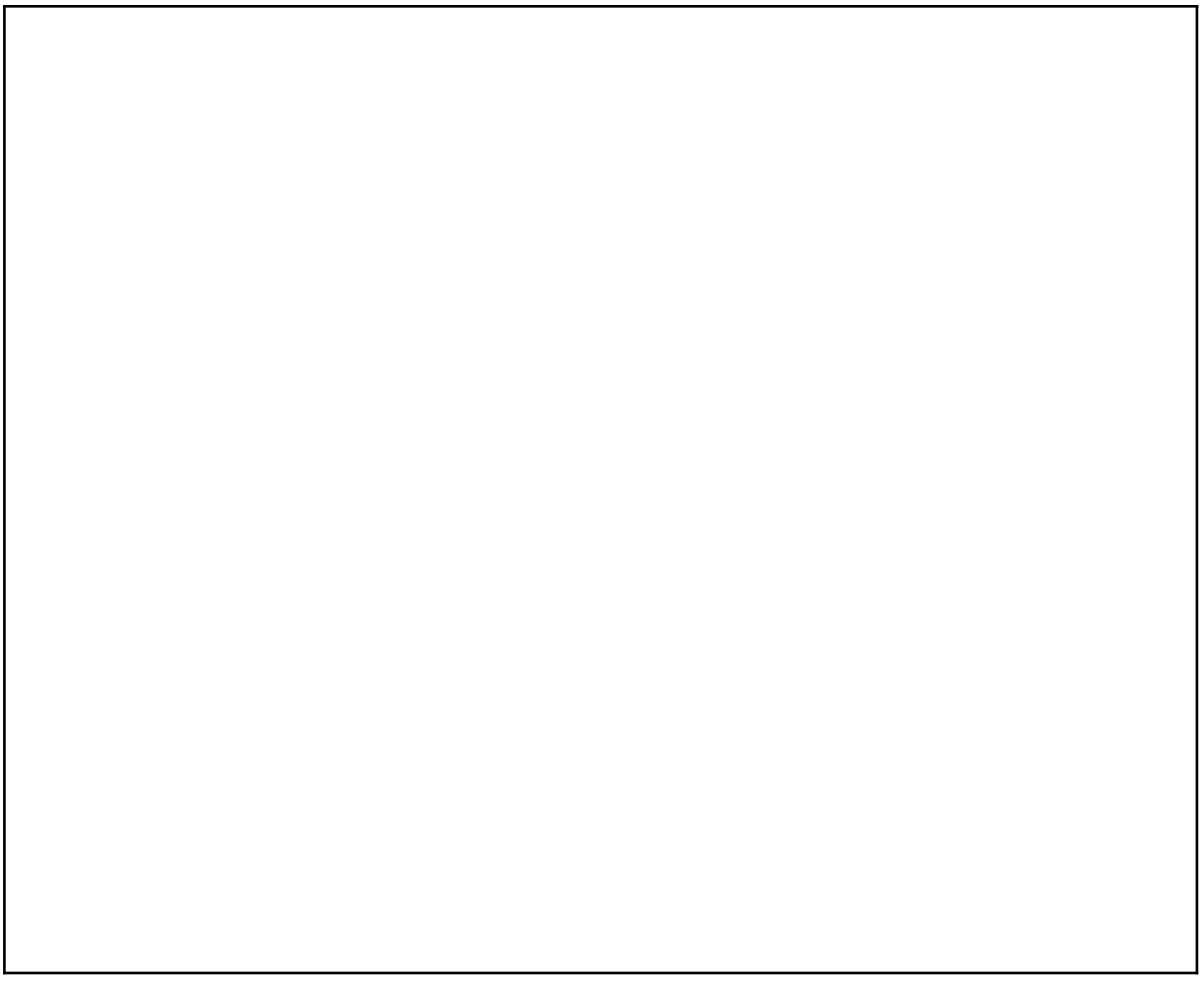
Code this method.  [5]



A tank commander needs to calculate the average damage rating of all currently available missiles in the Tank.

Write a Tank class method `getAverageRating` which will calculate and return the average damage rating of all Missiles in the Tank when the method is called.

Implement the `getAverageRating` method of the Tank class. It returns -1 if there are no Missiles available. [5]



Extension A Inheritance[1]

Part 1

A `Bombasticator` IS-A `Missile`. It has one additional property, which is an `potency` indicator (true or false). This is set by a parameter when a `Bombasticator` is constructed.

Design the entire `Bombasticator` class. [4]

[1] Attempt this if you have already studied Inheritance

Extension A Part 2

When a Bombasticator returns its DamageFactor it behaves similarly to a Missile but adds an extra 20% onto the returned value.

Using the concept of method overriding, design the getDamageFactor method of the Bombasticator class. [2]

Extension B Collections

A COLLECTION of Missiles exists, MISSILES.

Write an algorithm in pseudocode which will output a count of all Missiles which have an even damage factor number. [6]

Extension C HL Only (Topic 5)

A Queue of Tanks exists, T.

The length of the Queue is unknown.

When each Tank is dequeued from the Queue, it shoots all of its missiles and then the next Tank is dequeued.

Code this algorithm. [4]