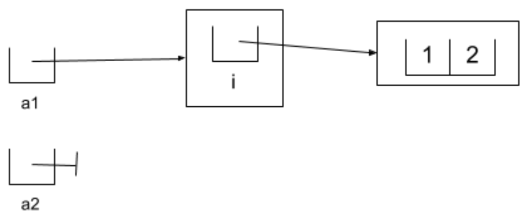
```
/*
 *   Suggested solutions exam 2020-08-19
 *
 */

   // -------------------------- 1 -------------------- (4p)
    void program() {
       Scanner sc = new Scanner(in);
       int k1, k2, k3;
       out.print("Första talet ? ");
       k1 = sc.nextInt();
       out.print("Andra talet ? ");
       k2 = sc.nextInt();
       out.print("Tredje talet ? ");
       k3 = sc.nextInt();

       int d = k2 - k1;    // First diff
       int f = (k3 - k2) - (k2 - k1); // Change in diff

       out.print("Talföljden: ");
       int k = k1;
       for (int i = 0; i < 10; i++) {
           out.print(k + ", ");
           k += d;  // Inc number
           d += f;  // Inc diff
       }
       out.println();
   }

   // -------------------------- 2 --------------------

   // No such question
 // ---------------- 3 ---------------------- (7p)

   int[] inBoth(int[] a1, int[] a2) {
       int[] tmp;
       int len = a1.length > a2.length ? a1.length : a2.length;
       tmp = new int[len];
       int k = 0;
       for (int i : a1) {
           for (int j : a2) {
               if (i == j && !contains(tmp, j)) {
                   tmp[k] = i;
                   k++;
               }
           }
       }
       int[] result = new int[k];
       for (int i = 0; i < result.length; i++) {
           result[i] = tmp[i];
       }
       return result;
   }

   boolean contains(int[] arr, int n) {
       for (int i : arr) {
           if (i == n) {
               return true;
           }
       }
       return false;
   }
```

```
// ---------------- 4 --------------------- (10p)

  int smallestMatrixWithSum(int[][] matrix, int sum) {
     for (int subSize = 1; subSize <= matrix.length; subSize++) {
        if (checkAllSubmatrices(matrix, subSize, sum) > 0) {
           return subSize;
        }
     }
     return -1;
  }


  int checkAllSubmatrices(int[][] matrix, int subSize, int sum) {
     for (int row = 0; row < matrix.length - subSize + 1; row++) {
        for (int col = 0; col < matrix.length - subSize + 1; col++) {
           if (sumMatrix(matrix, row, col, subSize) == sum) {
              return subSize;
           }
        }
     }
     return -1;
  }


  int sumMatrix(int[][] m, int row, int col, int size) {
     int sum = 0;
     for (int r = row; r < row + size; r++) {
        for (int c = col; c < col + size; c++) {
           sum = sum + m[r][c];
        }
     }
     return sum;
  }
```

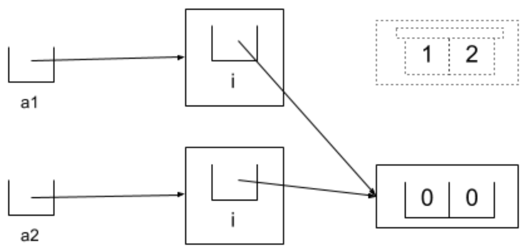
```
// ------------------------ 5 ------------------ (8p)

  boolean willHit(int startX, int startY, int endX, int endY, String walk) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < walk.length(); ) {  // NOTE: No i++ last
      char dir = walk.charAt(i);
      i++;
      while (i < walk.length() && Character.isDigit(walk.charAt(i))) {
        sb.append(walk.charAt(i));
        i++;
      }

      int step = Integer.parseInt(sb.toString());
      switch (dir) {
        case 'n':
          startY = startY + step;
          break;
        case 'e':
          startX = startX + step;
          break;
        case 's':
          startY = startY - step;
          break;
        case 'w':
          startX = startX - step;
          break;

      }
      sb.setLength(0);
    }
    return (startX == endX && startY == endY);
  }
```

// ---------------- 6 -------------------------------- (8p)

Before



After



```
 // -------------------- 7 ------------------------- (12p)

public class Bag {
        private final Random rand = new Random();
        private final List<Tile> tiles;

        public Bag(List<Tile> tiles) {
            this.tiles = tiles;
        }

        boolean isEmpty() {
            return tiles.isEmpty();
        }

        Tile getTile() {
            return tiles.remove(rand.nextInt(tiles.size()));
        }
    }


    public class Rack {
        public static final int MAX = 7;
        private final List<Tile> tiles = new ArrayList<>();

        boolean add(Tile tile) {
            if (tiles.size() < MAX) {
                return tiles.add(tile);
            } else {
                return false;
            }
        }

        Tile remove(Tile tile) {
            if (tiles.contains(tile)) {
                return tiles.remove(tiles.indexOf(tile));
            } else {
                return null;
            }
        }
    }

    public class Player {
        private final String name;
        private Rack rack = new Rack();

        public Player(String name) {
            this.name = name;
        }
    public Rack getRack() {
        return rack;
    }
}


public class Scrabble {

    private final Board board;
    private final List<Player> players = new ArrayList<>();
    private final Bag bag;
    private Player current;

    public Scrabble(Board board, Bag bag) {
        this.board = board;
        this.bag = bag;
    }


    boolean addToRack() {
        if (bag.isEmpty()) {
            return false;
        }
        Tile t = bag.getTile();
        return current.getRack().add(t);
    }

    boolean putOnBoard(Tile tile, int row, int col) {
        Tile t = current.getRack().remove(tile);
        if (t != null) {
            return board.put(tile, row, col);
        }
        return false;
    }
}
```

// --------------------- 8 --------------------------- (8p)

a) Utkrift

```
doIt A 5.0
doIt A 5.0
```

This has to do with overloading vs overriding. The method doIt is overloaded because the parameters differ. Overloaded methods are bound (decided which one to run)
at compile time, using the variable's static (declared type) So...

```
X x = new Y()  // Variable of type x reference y-object
x.doIt(5)      // Compiler tries to find method to call. No exact match in X
               // but a match if implicit type conversion of 5 to 5.0 (double)
               // Ok! That method is now fixed for execution during run.
               // If no matching method a compile error.
               // NOTE: Not calling exact matching method in y-object
               // (not overriding)
y.doIt(5.0)    // Will call inherited method from X, exact match at compile time.
```

b) Utskrift

```
ctor A
ctor B
ctor C
ctor A
ctor B
ctor C
```

This has to do with initialization. Before an object can be created the class must be loaded (from the *.class-file). Static variables are initialized at class loading, before any object created. So..

```
C c = new C()    // Must load class C before create object
                 // But C inherits B must load B
                 // B inherit A must load  A
                 // Start initialization of static variable c in B
                 // Create object of class C, call constructor C,
                 // must calls constructor B and B call constructor A.
                 // Constructor A prints out  and returns
                 // Constructor B prints out and returns
                 // Constructor C prints out and returns
                 // Finished with static variable
                 // Start creating objects referenced by variable c in the same
                 // manner as for the static variable
                 // Create object of class C, call constructor C,
// must calls constructor B and B call constructor A.
// Constructor A prints out  and returns
// Constructor B prints out and returns
// Constructor C prints out and returns
```