15. (a) *Award **[2 max].***
The signature defines the parameters (and their types);
The method name / identifier;
Allow either the return type or access modifier for a mark;

(b) *Award **[2 max].***
String is an immutable object / Java uses pass-by-value / does not use pass-by-reference;
It cannot simply have its value reassigned inside the method;
The **method** type must be `String`;
To **return** the **new value** for that String variable;

(c) *Award **[1 max].***
To output the addresses of **all** houses in a given city;

Note to examiners. Must imply all the houses that match, not just one.

(d) (i) *Award **[2 max]***
The length of the array may not correspond to the number of houses stored;
Causing a null pointer error/exception;
Since the loop may try to access the `getCity` method for a null entry;

(ii) *Award **[2 max.]***
Declare a variable `count` that stores the number of objects in `allHouses`;
Change loop condition to `j<count`;

Add a test `if(allHouses[i]!=null);`
Before testing `allHouses[i].getCity().equals(x) ;`

Use a while loop with a double condition;
`(i<allHouses.length)&&(allHouses[i]!=null) ;`

Initially fill the array with dummy objects;
So that no null entries will be encountered;

Change the array to an arrayList;
So that no null entries will be encountered / so that the array has the exact number of valid objects in it;

(e) *Award **[5 max]**.*
*Award **[1]** for correctly declaring a swap variable;*
*Award **[1]** for correct outer loop;*
*Award **[1]** for correct inner loop;*
*Award **[1]** for correctly testing prices (allow `.price`);*
*Award **[1]** for correct swap statements (ignore here any incorrect declaration of swap variable);*

Notes to examiners.
- *As there are many variations on sort routines you will need to carefully read through each response. You can ignore minor syntax errors in every algorithm answer.*
- *Note that is the House objects that are being swapped, not the prices. However, if prices are being swapped, allow FT for ta swap variable declared as a numeric type.*

Example answer 1

```
public void houseSort()              // basic bubblesort 1
{
  House swapHouse;
  for (int i=0; i< allHouses.length-1; i++ //accept
                                           //allHouses.length
  {
    for (int j=0; j< allHouses.length-1; j++)
    {
      if (allHouses[j].getPrice()>allHouses[j+1].getPrice())
      {
        swapHouse = allHouses[j];
        allHouses[j] = allHouses[j+1];
        allHouses[j+1] = swapHouse;
      }
    }
  }
}
```

Example answer 2

```
public void houseSort()               // basic bubblesort 2
{
  boolean isSorted = false;
  House houseTemp;
  while(!isSorted)
  {
    isSorted = true;
    for(int j=0;j<allHouses.length-1;j++)
    {
      if(allHouses[j].getPrice()>allHouses[j+1].getPrice())
      {
         houseTemp = allHouses[j];
         allHouses[j] = allHouses[j+1];
         allHouses[j+1] = houseTemp;
         isSorted=false;
       }
     }
   }
 }
```

Example answer 3

```
public void houseSort()           // basic selection sort 1
{
  House swapHouse;
  int smallest;
  for (int i=0; i< allHouses.length-1; i++)
  {
    smallest = i;
    for (int j=i+1; j< allHouses.length; j++)      // allow j =i
    {
      if (allHouses[j].getPrice()<allHouses[smallest].getPrice())
      {
        smallest = j;
      }
    }
      swapHouse = allHouses[smallest];
      allHouses[smallest] = allHouses[i];
      allHouses[i] = swapHouse;
  }
}
```

Example answer 4

```
public void houseSort()           // basic selection sort 2
{
    House swapHouse;
    for (int i=0; i< allHouses.length-1; i++)
      {
     for (int j=i+1; j< allHouses.length; j++)      // allow j = i
     {
       if (allHouses[i].getPrice()>allHouses[j].getPrice())
       {
         swapHouse = allHouses[i];
         allHouses[i] = allHouses[j];
         allHouses[j] = swapHouse;
       }
     }
   }
 }
```

(f) *Award **[7 max]**.*
*Award **[1]** for correct return type and return;*
*Award **[1]** for correct parameter, budget (allow* `allHouses` *to be passed as well);*
*Award **[1]** for correctly instantiating a result array;*
Award [1] for correctly using `houseSort`*;*
*Award **[1]** for while loop with one correct condition; // other loops could be used*
*Award **[1]** for while loop with two correct conditions in the right order; // note that some of these conditions may be implemented as IF statements;*
*Award **[2]** for assigning all 3* `House` *objects in the correct order (award **[1]** for good attempt at identifying the 3 objects);*

Example answer 1

```
public House[] selectThree(int budget)
{
  House[] result = new House[3];
  houseSort();                    // allow allHouses.houseSort()
  int i=0;
  while ((i<allHouses.length)&&(allHouses[i]!=null)
                             &&(allHouses[i].getPrice()<=budget))
  {
    i++;
  }
  i--;
  for (int a=0; a<=2; a++)
  {
    result[a] = allHouses[i-2+a];
  }
  return result;
}
```

Example answer 2

```
public House[] selectThree(int budget)
{
  House[] result = new House[3];
  houseSort();
  int i=0;
  while ((i<allHouses.length)&&(allHouses[i]!=null)
                             &&(allHouses[i].getPrice()<=budget))
  {
    i++;
  }
  i--;
  result[0] = allHouses[i-2];
  result[1] = allHouses[i-1];
  result[2] = allHouses[i];
  return result;
}
```

Example answer 3

```
public House[] selectThree(int budget)
{
  House[] result = new House[3];
  houseSort();
  int index = 2;
  for(int i=allHouses.length-1; i>=0 && index>=0; i--)
  {
    if(allHouses[i]!=null && allHouses[i].getPrice()<=budget)
         result[index--] = allHouses[i];
  }
  return result;
}
```