HashMap I

Java provides a HashMap class (and a Hashtable class)

Recapitulation: A hash map consists of <key/value> pairs, stored in an array. The key is transformed into an array index with a hash function, enabling direct access of the value.

To make it work (for the provided Java HashMap class), the key object must provide a hashCode() method (returning a 32-bit integer, the hash value).

The Hash Map translates this hash value into an array index, using modular hashing (what the heck is that again?)

Solution: (you don't have to do that, HashMap does it)

Divide the hash value by the array size and use the reminder

Hash Map II

Good news: Java provides a hashCode() method for all ist classes.

Bad news: For your own classes you must provide your own hashCode(). To be precise: You must overwrite the hashCode() from the Object class.

If you don’t overwrite this method, the objects hashCode() is used, which uses its address as hash value.

Why is that bad? Think about it!

Only identical objects have the same hash value, whereas normally you would want equal objects to have the same hash value.
(You know the difference between identical and equal, don’t you?)

BTW, you **MUST** also overwrite the equal() method, hashCode() and equal() work together.

Task: write hashCode() and equal() for a 3-dimensional Point class:

Class Point3D { int x, y, z; }

Recursion Exercises

1) What is the output of rec1(5):

```
public static String rec1(int n) {
     if (n <= 0) return "";
     return n + rec1(n-1) + n;
}
```

2) What is the output of rec2(4):

```
public static String rec2(int n) {
     if (n <= 0) return "";
     return n + rec2(n-1) + rec2(n-2);
}
```

3) What is the output of rec3(6):

```
public static String rec3(int n) {
     String s = rec3(n-2) + rec3(n-3) + "A";
     if (n <= 0) return "";
     return s;
}
```