Classes and Objects in Java

Almost everything is a class in Java, even Strings and Arrays. So when you create an array variable or a String variable what you are creating is an object Array and String class.
Some people do not consider Java as a Purely Object Oriented Language due to the presence of primitive data types like int, boolean etc. (Java supports 8 primitive data types in total). Though the language provides equivalent class for each primitive data type (called wrapper classes e.g. Integer for int) but, the fact that it supports primitive types doesn't allow Java to be labelled as Purely Object Oriented language.

Object Memory Allocation

Objects are allocated memory in heap. Understanding memory allocation will help you to understand why certain code pertaining to objects behaves.

Objects are allocated memory as soon as we use the **new operator.** However, if we have just declared an object of a certain class and haven't used new to initialize it then no object is created in memory.

The reference variables (objects) holds the address (often called **a reference** in java terminology) of the actual object value stored in heap memory. We can understand it better through a memory diagram consider following code segment.

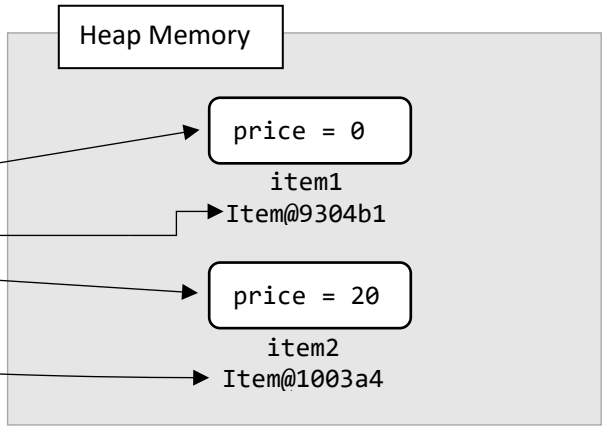
Class:

```
1   class Item
2   {
3      int price;
4   }
```

Example 1.1: Object Memory Allocation

Code:

```
1   Item item1 = new Item();
2
3   Item item2 = new Item();
4   item2.price = 20;
5
6   System.out.println(item1);
7   System.out.println(item2);
```



Output:

```
Item@9304b1
Item@1003a4
```

Explanation: As you can see from the diagram we have created two objects of item class in heap memory.

The **new operator** creates the objects in memory and returns the reference of newly constructed objects.

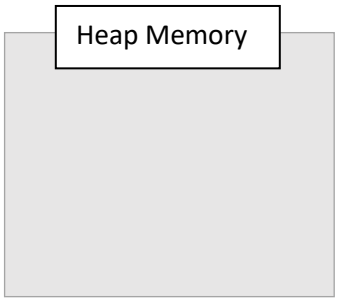
The **reference values** are stored in item1 and item2. All object variables are also called reference variables because they hold the reference.

Note: Directly printing an object will print its reference value.

References are of this form:

ClassName@HexadecimalCode

Example 1.2: Object Memory Allocation without new



Code:

```
1   Item item3;
2   Item item4;
```

Explanation: The program will not create any object in memory. The code merely declares two reference variables item3 and item4. Actual memory is allocated by new operator only.

Example 1.3: Common errors

Code:

```
1   Item item5;
2   System.out.println(item5);
```

Output:

```
Error in Line 2
```

Explanation: The above program will generate compile time error in line 2 because item5 variable isn't initialized.

Example 1.4: null reference

Code:

```
1   Item item6 = null;
2   System.out.println(item6);
3   System.out.println(item6.price);
```

Output:

```
null
Exception in line 3: NullPointerException
```

Explanation: item6 holds null reference. This means it does not point to any valid object in memory. Directly, printing item6 will give null. However, if we try to call any method or access any variable through null reference we'll get NullPointerException.

Memory: Stack vs Heap

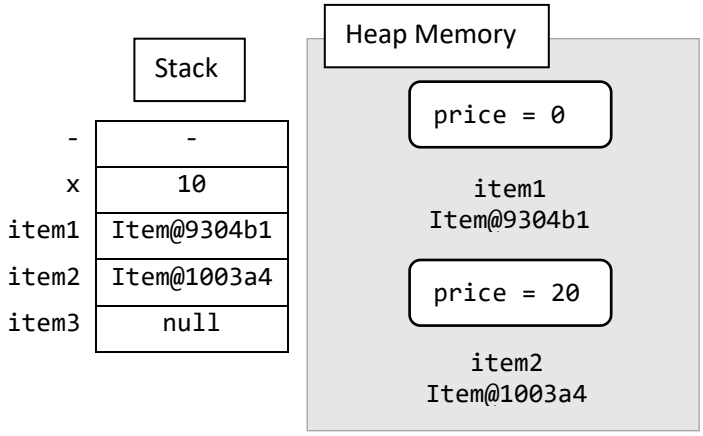
Java uses two types of memory areas:

- Stack – Holds primitive data types (int, boolean, etc) and reference values.
- Heap – Holds object values.

Example 2: Stack vs Heap

Code:

```
1   int x = 10;
2   Item item1 = new Item();
3
4   Item item2 = new Item();
5   item2.price = 20;
6   Item item3 = null;
```



Explanation: As you can see from the diagram above the actual objects are created in the heap memory. However, all the primitive variables and reference variables (item1 and item2) are stored on stack.

Note: item1 and item2 are reference variables. They hold the reference values in stack. The actual object resides in heap memory.

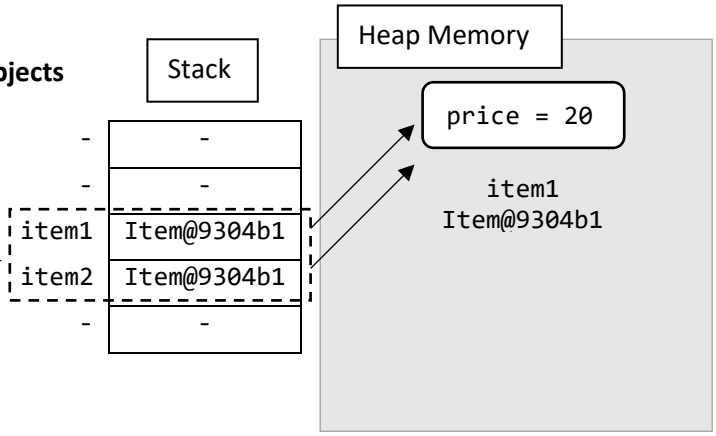
Object Assignments

We use assignment operator to copy value from one primitive data type variable's value to another. However, in case of objects assignment works a little differently.

Example 3.1: Assignment Operator (=) with objects

Code:

```
1   Item item1 = new Item();
2   item1.price = 10;
3   Item item2 = item1;
4   item2.price = 20;
5   System.out.println(item1.price);
6   System.out.println(item2.price);
```



Output:

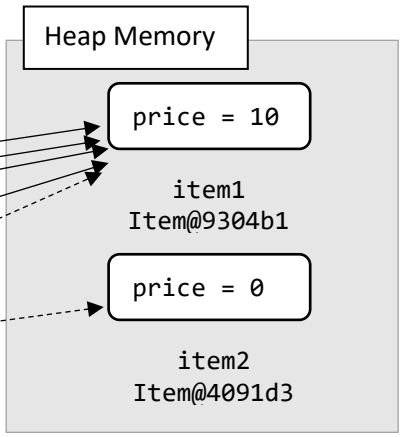
```
20
20
```

Explanation: Assignment operator in case of reference variable copies the reference value from one variable to another. Since both the reference variables contain the same reference value they both point to same object in memory. Hence, any change made through any reference variable will reflect in the other.

Example 3.2: More on object assignment

Code:

```
1   Item item1 = new Item();
2   item1.price = 10;
3   Item item2 = item1;
4   Item item3 = item2;
5
6   System.out.println(item2.price);
7
8   item2 = new Item();
9   System.out.println(item3.price);
```



Output:

```
10
10
```

Explanation: Due to line 3 and 4 all reference variables - item1, item2 and item3 point to same object in memory. Line 8 creates a new object in memory using new keyword hence item2 starts pointing to this new object. Line 8 does not affect item1 and item3. They still point to old objects.

Object Equality: '=='

We frequently compare two objects of same class to ascertain whether they are same or not. Double equal operator (==) isn't enough to check whether two objects are same or not. '==' merely checks if they point to the same object in memory i.e. if their reference values are same or not. But, cases where we want to check whether the 2 objects of same class contain same value for each field are not handled by '=='.

Remember: For objects '==' checks only reference value not the actual value of objects.

Example 4: Demonstrating where '==' operator fails

Code:

```
1   Item item1 = new Item();
2   item1.price = 10;
```

```
3   Item item2 = item1;
4   Item item3 = new Item();
5   item3.price = 10;
6
7   System.out.println(item1==item2);
8   System.out.println(item1==item3);
```

Output:

```
true
false
```

Explanation: item1 and item2 both point to same objects hence '==' gives true for them but, false for item1 and item3. However, as you can see from the code both item1 and item3 contains same value for their price variable so they are structurally same but, we get false.

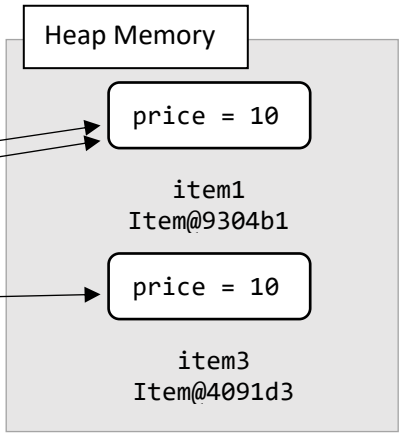
Object Duplicates

A simple assignment do not create a new object in memory instead it merely copies the reference value. This creates problems when we need a duplicate (which is structurally same and having its own memory). A simple assignment seems insufficient for this task. The best way is to allocate new memory and copy each class field's values from one object to another. There are built-in methods in java that do this task for you. But, to understand the whole concept will do it by copying each field one by one.

Example 5: Creating object duplicates

Code:

```
1   Item item1 = new Item();
2   item1.price = 10;
3   Item item2 = item1;
4   Item item3 = new item();
5   item3.price = item1.price;
6
7   System.out.println(item1==item2);
8   System.out.println(item1==item3);
9   System.out.println(item1.price==item3.price);
```



Output:

```
true
false
true
```

Explanation: item1 and item2 both point to same objects hence '==' gives true for them. item3 and item1 on the other hand refer to two different objects in memory. However they are structurally same (both have price variable with same value). So we can say that item3 and item1 are duplicates of one another however, item1 and item2 aren't.

Problem 1	How are objects allocated memory? Explain with example.

Problem 2	Explain working of heap memory and stack with an example.

Problem 3	What are reference variables? Explain with example.

Problem 4	How many objects will be created in each of the following segments? Explain with a diagram for each.

Class

```
1  class Point
2  {
3     int x;
4     int y;
5  }
```

Code 4.1

```
1  Point p1 = new Point();
2  p1.x = 10;
```

Code 4.2

```
1  Point p1, p2, p3;
2  p1 = new Point();
```

Code 4.3

```
1  Point p1 = new Point();
2  Point p2 = p1;
3  Point p3 = p1;
```

Problem 5	Give output for each of the following code segments. Explain your answer using memory diagram.

Code 5.1

```
1  Point p = new Point();
2  System.out.println(p);
3  System.out.println(p.x);
4  System.out.println(p.y);
```

Code 5.2

```
1  Point p1 = null;
```

```
2  System.out.println(p1);
3  System.out.println(p1.x);
```

Code 5.3

```
1  Point p1 = new Point();
2  p1.x = 10;
3  p1.y = 20;
4  Point p2 = p1;
5  p2.x = 30;
6  p2.y = 40;
7  System.out.println("P1::" + p1.x + "," + p1.y);
8  System.out.println("P2::" + p2.x + "," + p2.y);
```

Code 5.4

```
1  Point p1 = new Point();
2  p1.x = 10;
3  Point p2 = p1;
4  p2.x = 30;
5  Point p3 = p2;
6  p3.x = 60;
7  System.out.println(p1.x);
8  System.out.println(p2.x);
9  System.out.println(p3.x);
```

Code 5.5

```
1  Point p1 = new Point();
2  Point p2 = p1;
3  p2 = new Point();
4  Point p3 = p2;
5  System.out.println(p1==p2);
6  System.out.println(p2==p3);
```

Code 5.6

```
1  Point p1 = new Point();
2  p1.x = 10;
3  Point p2 = p1;
4  Point p3 = new Point();
5  p3.x = p2.x;
6  System.out.println(p1==p2);
7  System.out.println(p1==p3);
8  System.out.println(p1.x==p3.x);
```

Problem 6	**How to create a duplicate of an object. Explain with example.**