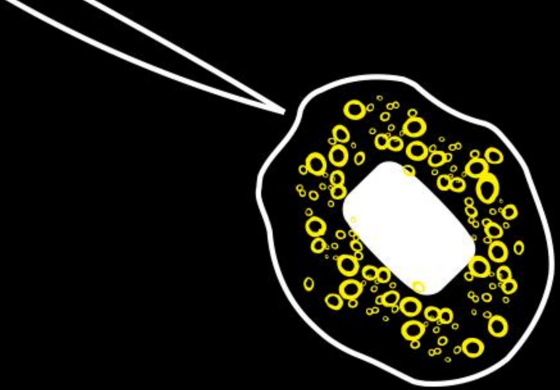
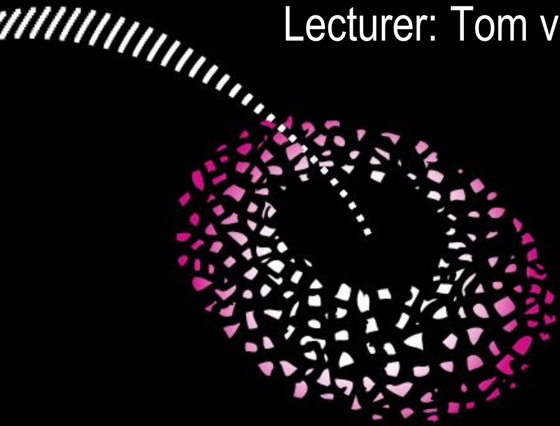
UNIVERSITY OF TWENTE.

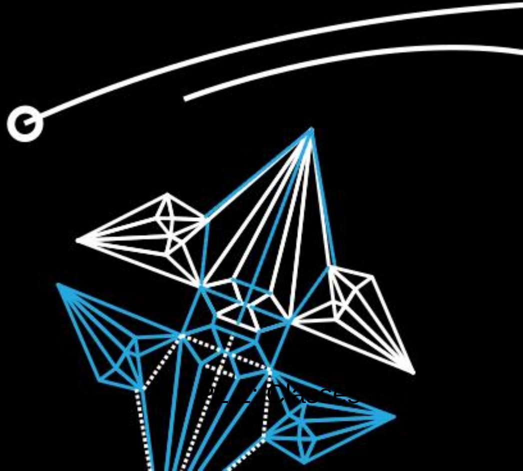


Objects in Memory

Topic of Software Systems (TCS module 2)

Lecturer: Tom van Dijk





OBJECTS IN MEMORY

MEMORY

Just a very long sequence of 0s and 1s

JAVA DATA TYPES

Primitive / reference types encoded as 0s and 1s

OBJECTS IN MEMORY

Primitive data types:

- boolean: 1 bit
- byte: 8 bits
- short: 2 bytes (16 bits)
- int: 4 bytes (32 bits)
- long: 8 bytes (64 bits)
- float: 4 bytes
- double: 8 bytes

OBJECTS IN MEMORY

Reference data types: (sometimes called pointers)

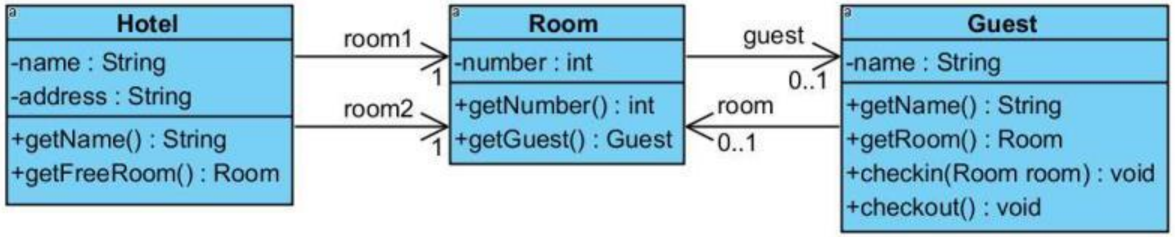
- Simply the location in the memory: typically 8 bytes (64 bits)
- (also called the memory address)

Composite data types like objects:

- All fields of the object in sequence
- Size: sum of field sizes + some overhead

EXAMPLE

Example classes for a hotel application:



EXAMPLE

```java
public class Room {
    private int number;
    private Guest guest;

    /*  Constructor
        Does not initialise guest attribute
    */
    public Room(int number) {
        this.number = number;
    }

    // to be continued
}
```

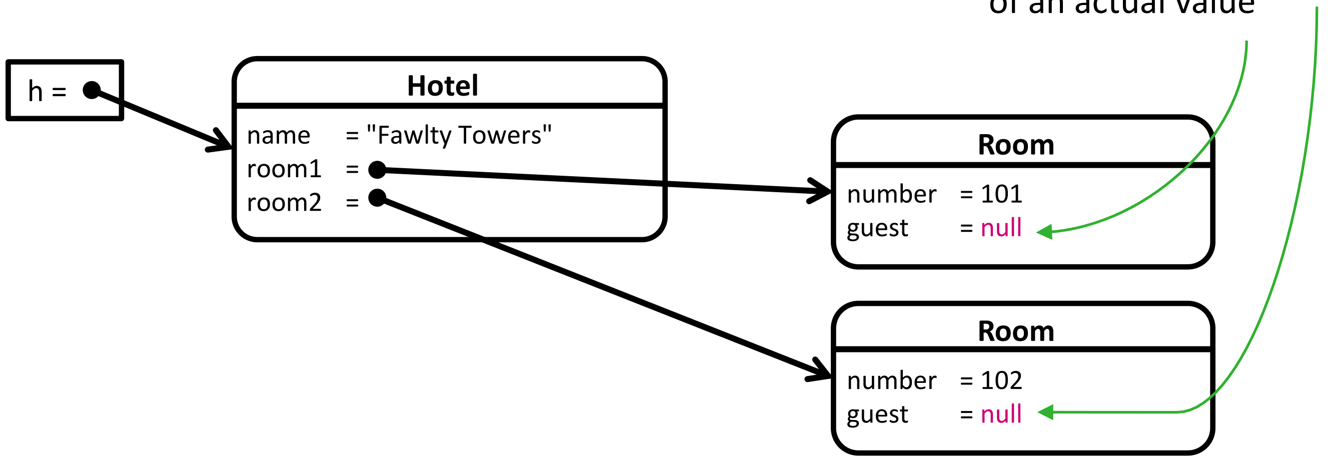
```java
public class Hotel {
    private String name;
    private Room room1;
    private Room room2;

    public Hotel(String name) {
        this.name = name;
        room1 = new Room(101); // constructor call
        room2 = new Room(102); // constructor call
    }

    // more stuff
}
```

EXAMPLE

The piece of code results in the creation of the following structure

```
Hotel h = new Hotel("Fawlty Towers");
```

null stands for the absence of an actual value

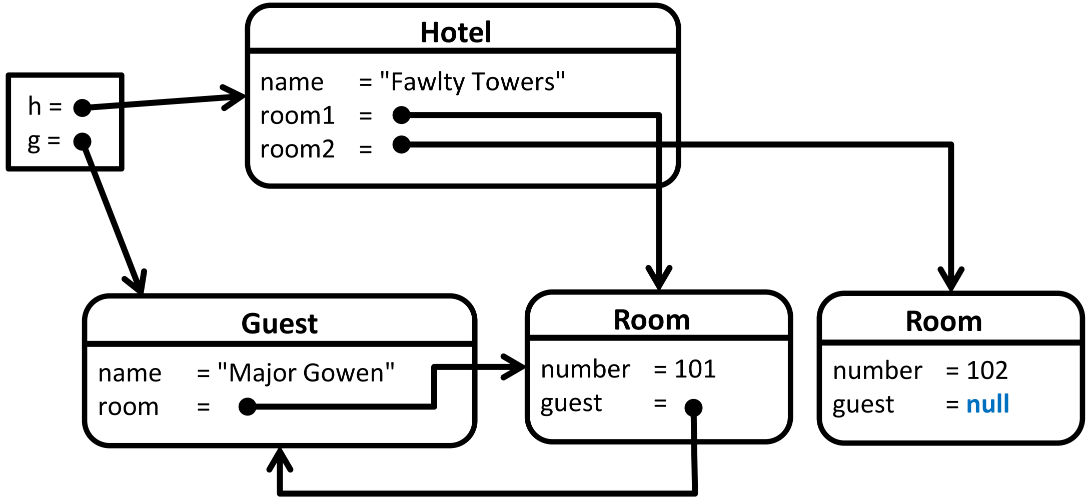


UNIVERSITY OF TWENTE.

EXAMPLE

- Objects can point back and forth
- There can be multiple pointers to the same object

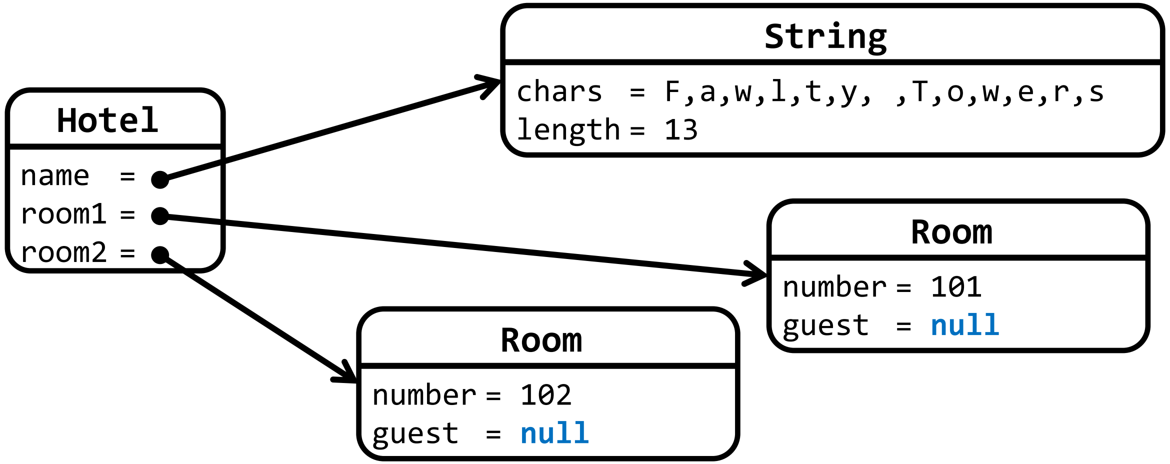
```
Hotel h = new Hotel("Fawlty Towers");
Guest g = new Guest("Major Gowen");
g.checkin(h.getFreeRoom());
```



STRINGS

We have actually omitted something

- Strings are also classes, and hence references to composite types
- `String` variables are pointers/references to `String` objects



UNIVERSITY OF TWENTE.

EQUALS VS ==

equals vs ==

- All Java objects have a method equals
- == compares the reference: true if same object
- equals (usually) compares the fields: true if same content

Primitive types

- have only ==

UNIVERSITY OF TWENTE.

CONCLUSION

IN MEMORY:

- Primitive types are simply the value
- Reference types point to a location in memory
- Composite types are the fields (plus overhead)

Use == to compare references, equals to compare content