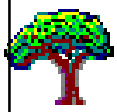
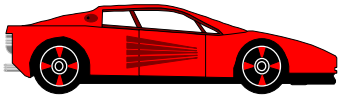
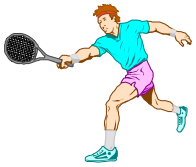
Objects

- We are all familiar with the idea of an object. We are surrounded by them: cars, books, people, houses, cats, etc.
- Objects have *attributes*, e.g. colour, size, age, name.



- Objects also have *behaviour*. They can do things, e.g. grow, breathe, run, stop.





Attributes and Behaviour

•	Attributes	Behaviour
Car	Registration number Make Model Year Colour	Start Stop Turn Accelerate
Cat	Name Breed Colour Age Weight Registration number	Sleep Eat Purr Climb Scratch

Abstraction and Models

- In a program, we choose the attributes and behaviours that are relevant to the problem we are trying to solve.
- This process of selecting some aspects of the objects and ignoring the irrelevant ones is called *abstraction*. We create a *model* that is a simplification of the real situation, strictly relevant to the problem to be solved.

Simulation

- When we write a program, we are often simulating something that happens in real life. The models we create in our program correspond to real life objects, e.g. customers, invoices, receipts.
- Sometimes the simulation is more explicit, e.g. a program that models a set of traffic lights to work out the optimum times for changing them to keep the traffic flowing smoothly.
- The object-oriented style (or paradigm) of programming is very well suited to simulations.

Object-Oriented Programs

- The object-oriented programming paradigm uses objects to model a situation. A program can be written in a non-object-oriented way, but the object-oriented style is increasingly popular for three main reasons:

- programs are becoming more and more complex, and the OO style handles this best
- the OO style makes it easy to re-use existing programs or parts of them
- the OO style makes programs much easier to maintain.

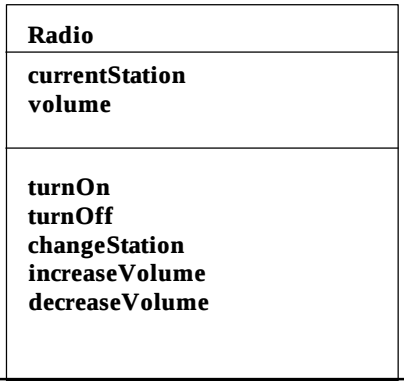
Classification

- We classify objects into groups according to their common attributes and behaviour.
- Let's say we want to classify objects into balls and not-balls. What are the common attributes and behaviour of something we would classify as a ball? How would you recognise a ball if you saw one?



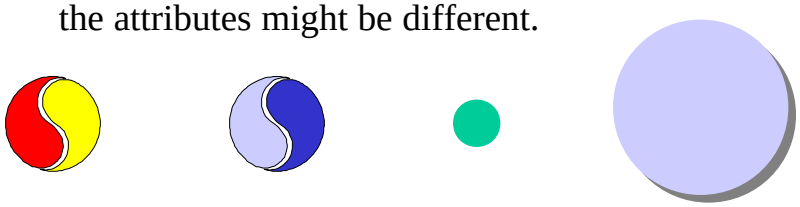
Classes

- In a computer program, we use *classes* to describe similar objects. A *class* is a description of what an object would look like if we had one. It is based on common attributes and behaviour of objects.



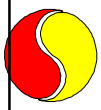
Instances of Classes

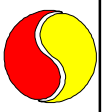
- A class is a template that describes what an object would look like if we had one.
- We *instantiate* the class to create an *instance*, i.e. an object of that class.
- The object is an instance of the class.
- We can have many instances of the same class. Each object has the same attributes and the same behaviour. However, the values of the attributes might be different.



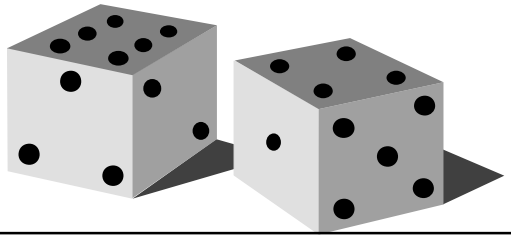
Identity

- We can have several different objects that have the same attributes and behaviour (even the same state) but are still different objects.





- An object has *identity*. We can give it a name to distinguish it from the others (e.g. ball1, ball2).



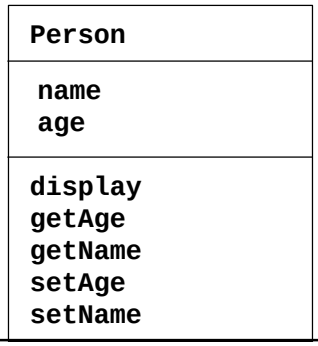
State of an Object

- The *state* of a particular object is the values of its attributes at any particular moment. For example, a cat might have the following state:
- Name: Max
- Breed: American Shorthair
- Colour: Black, brown & white
- Age: 3 years, 4 months, 8 days
- Weight: 5.1 kg
- Registration number: 21232322

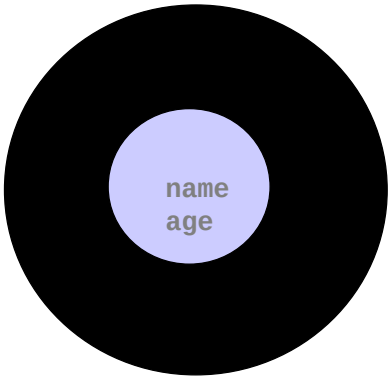
- Some of these attributes change constantly. Others stay the same for long periods or forever.

A *Person* Class

- Let's imagine we need a model of a person to solve some problem. For the purposes of this problem, it is enough to let a person have just a name and an age.



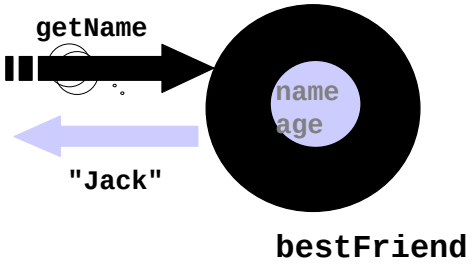
An Object Hides Its Attributes



If you want to find out the value of an attribute of an object, you have to ask the object to tell you.

Message Passing

- To get an object to do something it knows how to do, you have to send it a message requesting that behaviour.



Sending a Message to an Object

- There are 3 parts in a message sent to an object:
- the name of the object that is the receiver
- the action that the receiver is requested to take
- in parentheses, any extra information the receiver needs to know.
- In Java, the *syntax* for asking the person called `bestFriend` to tell you its name would be:

  – `bestFriend.getName();`

Passing Information in a Message

- When an object needs to know some extra information in order to do what you want, you can pass it that information with the message.
- If you decide that Jack is no longer your best friend but Sally now is, and want the object `bestFriend` to change its name to "Sally", you need to send it a message that tells it to change its name, and also what to change it to.
- The extra information is called *arguments* or *parameters*.

  - `bestFriend.setName("Sally");`