## **Procedural Programming**

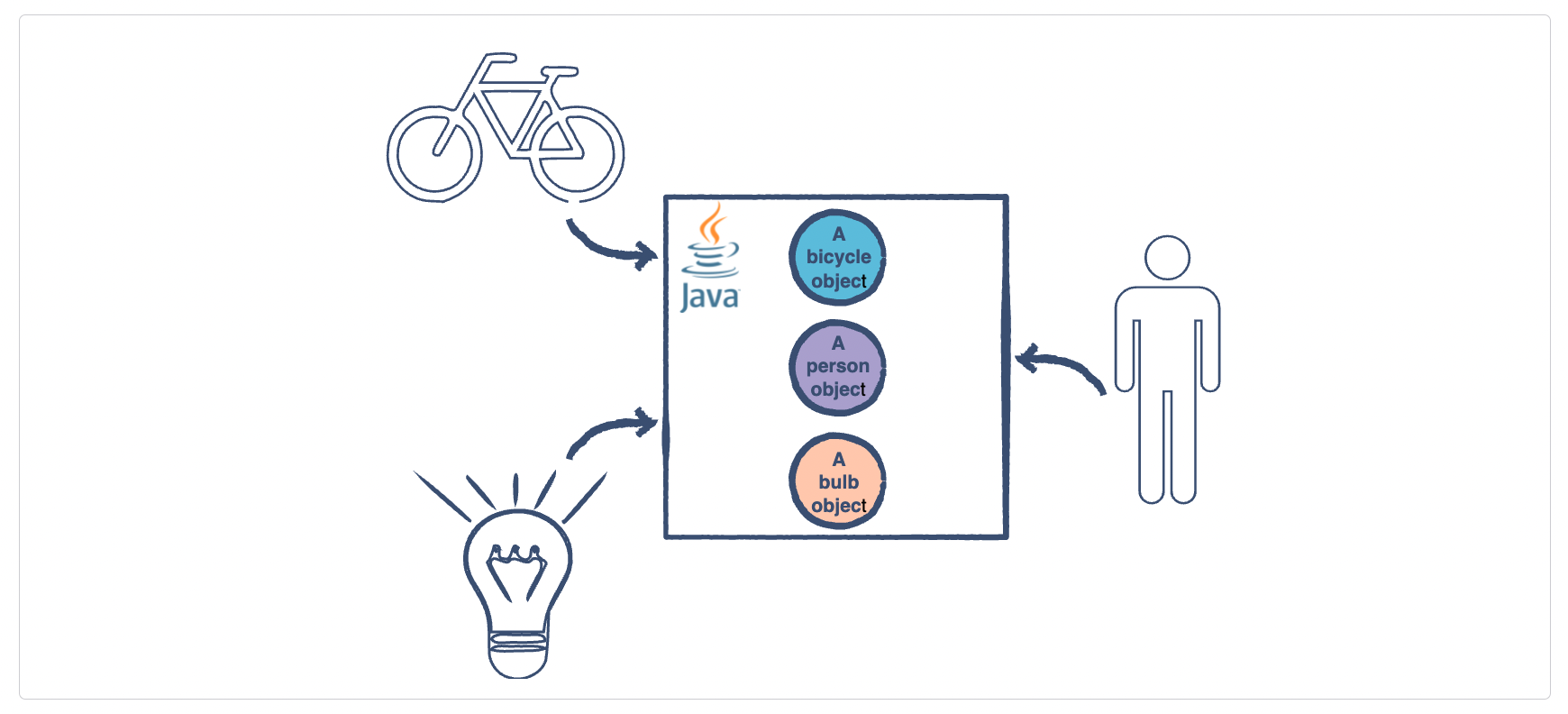
In procedural programming, **a program is divided into smaller parts called methods**. These **methods are the basic entities** used in this technique. The **focal point of the procedural programming technique** is to **use methods for code reusability**. However, the implementation of a complex real-world scenario becomes a difficult task using this programming paradigm.

## **Object-Oriented Programming**

**Object-oriented programming** also referred to as **OOP is a** **programming paradigm** that includes or **relies on the concept of classes and objects**. The **basic entities in object-oriented programming** are **classes and objects**.

**The basic idea of OOP is to divide a complex program into a bunch of objects talking to each other**.

**Objects** in a program frequently **represent real-world objects**.



**Many other objects serve application logic and have no direct real-world parallel objects** that manage authentication, templating, request handling, or any of the other myriad features needed for a working application.

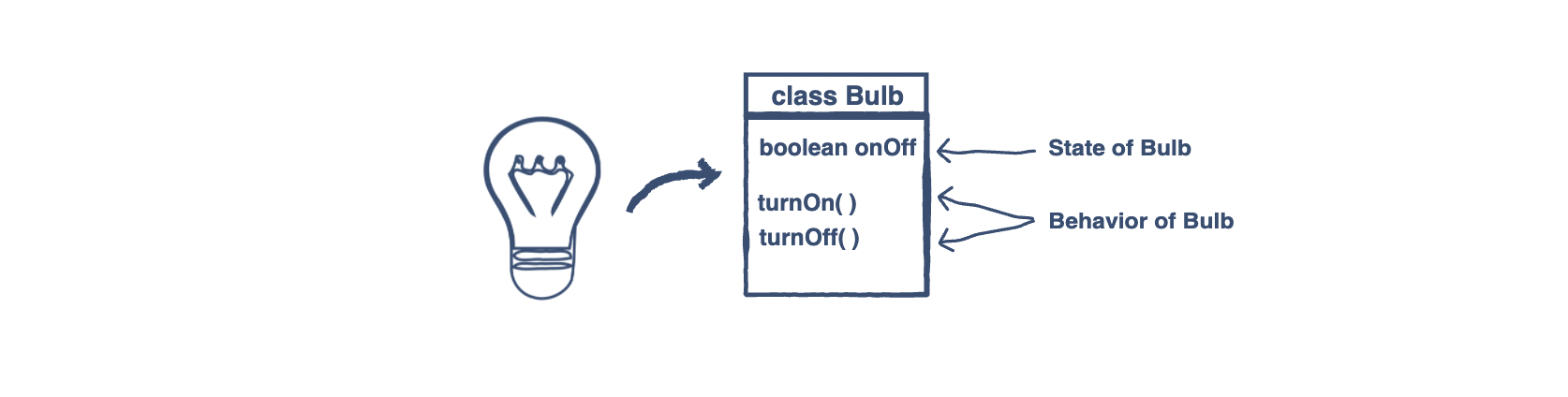
## **Anatomy of Objects and Classes**

**Objects** may **contain data in the form of fields (variables)** and **methods to operate on that data**. Take the example of a light bulb. It has a state i.e. either it is on or off. It also has a behavior i.e. when you turn it on it lights up and when turned off, it stops spreading light. To conclude this one can say:

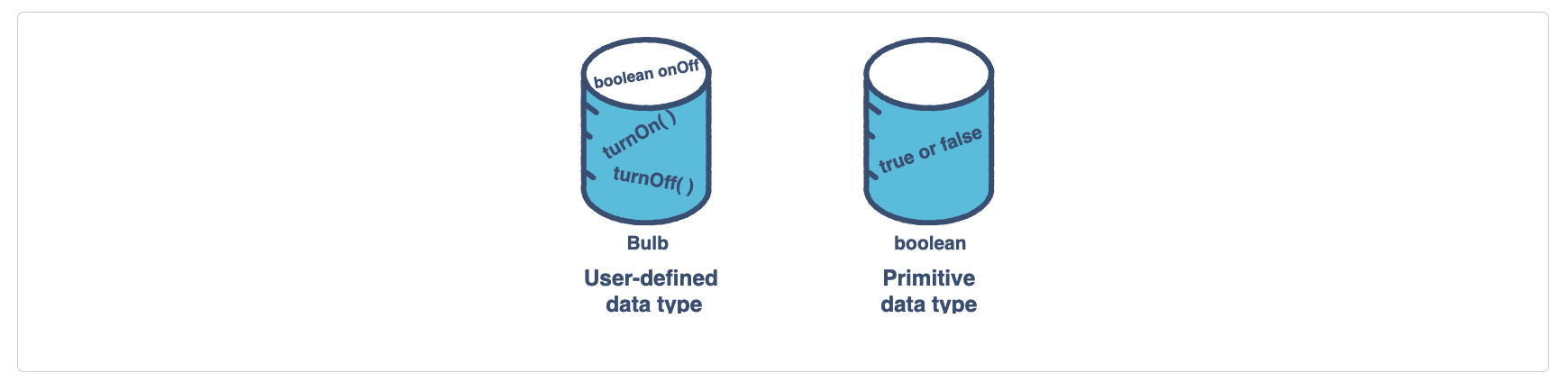
**Objects have state and behavior**.

**A** **Class** **can be thought of as a** **blueprint for creating objects**.

The below illustration shows what a **LightBulb** class should look like.



From the above illustration, it can be concluded that the **state of the objects** is generally **modeled using variables in a class** and the **behavior** is **modeled by implementing the methods**.



From the above discussion, it can be inferred that **classes are user-defined data types** **implemented using the primitive data types** e.g. **boolean, int, char,** etc.

**Abstraction**

* An **abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects** and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.
* **Abstraction means to focus on the essential features of an element or object in OOP, ignoring its extraneous or accidental properties**. The essential features are relative to the context in which the object is being used.

**Encapsulation**

* **Encapsulation is the process of binding both attributes and methods together within a class**.
* **Through encapsulation, the internal details of a class can be hidden from the outside**.
* The class has methods that provide user interfaces by which the services provided by the class may be used.

**Inheritance**

* **Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another**.
* **With the use of inheritance, the information is made manageable in a hierarchical order**.
* The class which inherits the properties of others is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

**Polymorphism**

* **Polymorphism is the ability of an object to take on many forms**.
* The most common use of polymorphism in OOP occurs when a **parent class reference is used to refer to a child class object**.
* Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

**Modularity**

* **Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules**.
* Modularity is the process of decomposing a problem (program) into a set of modules so as to reduce the overall complexity of the problem.

**Hierarchy**

* **Hierarchy is the ranking or order of abstraction**.
* Through hierarchy, **a system can be made up of interrelated subsystems**, which can have their own subsystems and so on until the smallest level components are reached.
* It uses the principle of **divide and conquer**.
* **Hierarchy allows code re-usability**.

The two types of hierarchies in OOA are:

* **ISA hierarchy**:
  + It **defines the hierarchical relationship in inheritance**, whereby **from a super-class, a number of subclasses may be derived** which may again have subclasses, so on.
  + For example, if we derive a class Rose from a class Flower, we can say that a rose isa flower.
* **PARTOF hierarchy**:
  + It **defines the hierarchical relationship in aggregation** by which **a class may be composed of other classes**.
  + For example, a flower is composed of sepals, petals, stamens, and carpel. It can be said that a petal is a part of a flower.

**Typing**

* According to the theories of abstract data type, **a type is a characterization of a set of elements**. In OOP, **a class is visualized as a type having properties distinct from any other type**.
* Typing is the **enforcement of the notion that an object is an instance of a single class or type**. It also enforces that objects of different types may not be generally interchanged, and can be interchanged only in a very restricted manner if absolutely required to do so.

The two types of typing are:

* **Strong Typing**:
  + Here, the **operation on an object is checked at the time of compilation**.
* **Weak Typing**:
  + Here, messages may be sent to any class. The **operation is checked only at the time of execution**.