**Class**

In the real world, we often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that the bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created.

Java classes contain fields and methods. A field is like a C++ data member, and a method is like a C++ member function. In other words, a class is a collection of object behavior and features or a class is a user defined data type.

**Java Class Library**

Java class library is a set of dynamically loadable libraries that Java applications can call at run time. Because the Java Platform is not dependent on any specific operating system, applications cannot rely on any of the existing libraries. Instead, the Java Platform provides a comprehensive set of standard class libraries, containing much of the same reusable functions commonly found in modern operating systems.

**Fields, Methods and Access Levels**

Each field and method has an access level:

* private: accessible only in that class
* package: accessible only in that package
* protected: accessible only in that package and in all subclasses of that class
* public: accessible everywhere in that class which is available

Similarly, each class has one of two possible access levels:

* package: class objects can only be declared and manipulated by code in that package
* public: class objects can be declared and manipulated by code in any package

**Simple Example Class**

class List

{

Data members;

Member functions()

public static void main(String args[])

{

Class Object();

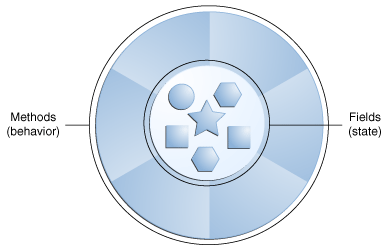
}

}

**Object**

Java is an object-oriented programming language. But what are objects? An object is a self-contained entity which has its own private collection of properties i.e. data and methods that encapsulate functionality into a reusable and dynamically loaded structure. After a class definition has been created as a prototype, it can be used as a template for creating new classes that add functionality. Objects are programming units of a particular class. Dynamic loading implies that applications can request new objects of a particular class to be supplied on an 'as needed' basis. Objects provide the extremely useful benefit of reusable code that minimizes development time.

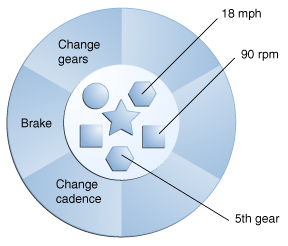
Objects are key to understand object-oriented technology. Real-world objects share two characteristics: They all have state and behavior. Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.



A software object.

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages). Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication. Hiding internal state and requiring all interaction to be performed through an object's methods is known as data encapsulation-- a fundamental principle of object-oriented programming.

For example, consider a bicycle:



A bicycle modeled as a software object.

By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it. For example, if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

Bundling code into individual software objects provides a number of benefits, including:

* Modularity: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
* Information-hiding: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
* Code re-use: If an object already exists, we can use that object in our program. This allows specialists to implement/test/debug complex, task-specific objects, which we can then trust to run in our own code.
* Plug ability and debugging ease: If a particular object turns out to be problematic, we can simply remove it from our application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, we replace it, not the entire machine.