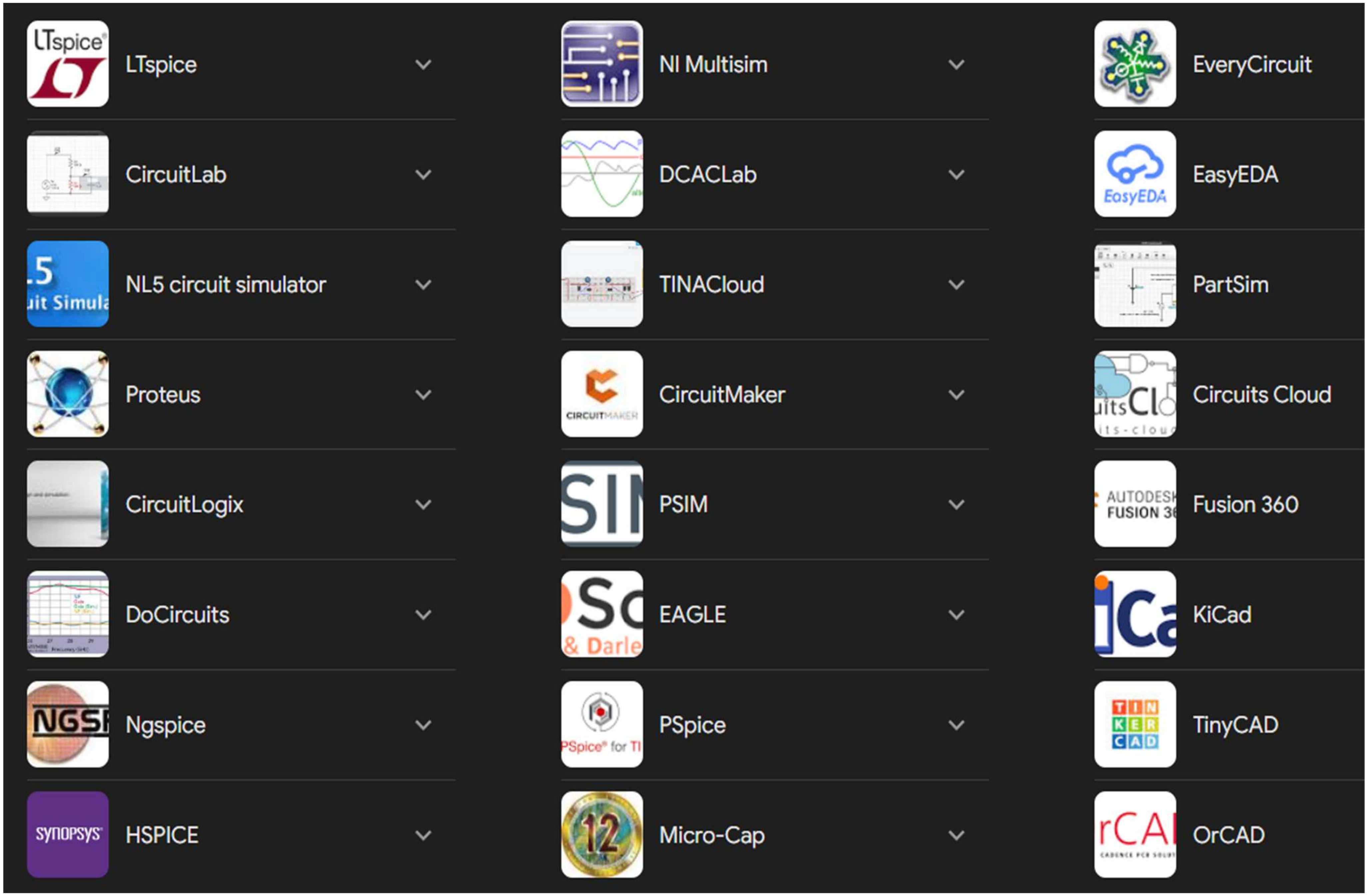
Object Oriented Programming

Prepared by

Dr. Rajesh Kumar Ojha
Asst. Prof., CSE, Silicon University



LTspice	NI Multisim	EveryCircuit
CircuitLab	DCACLab	EasyEDA
NL5 circuit simulator	TINACloud	PartSim
Proteus	CircuitMaker	Circuits Cloud
CircuitLogix	PSIM	Fusion 360
DoCircuits	EAGLE	KiCad
Ngspice	PSpice	TinyCAD
HSPICE	Micro-Cap	OrCAD

What is OOPs?

- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.
- It simplifies the software development and maintenance by providing some concepts:
- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

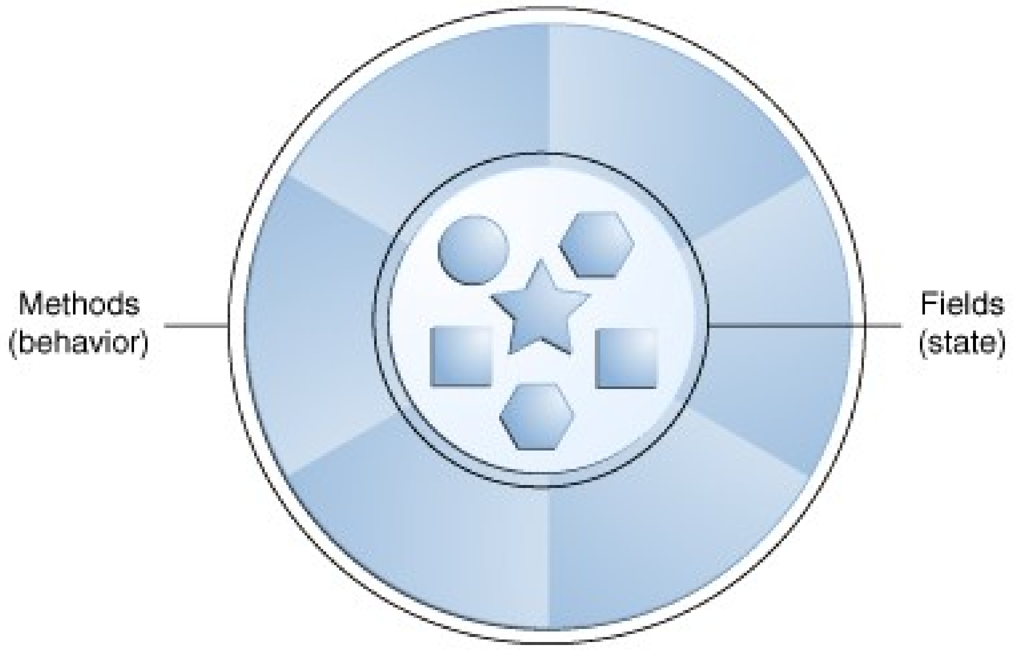
Object?

- Objects are key to understanding *object-oriented* technology. Look around right now and you'll find many examples of real-world objects: your dog, your desk, your television set, your bicycle.
- 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.
- Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes).

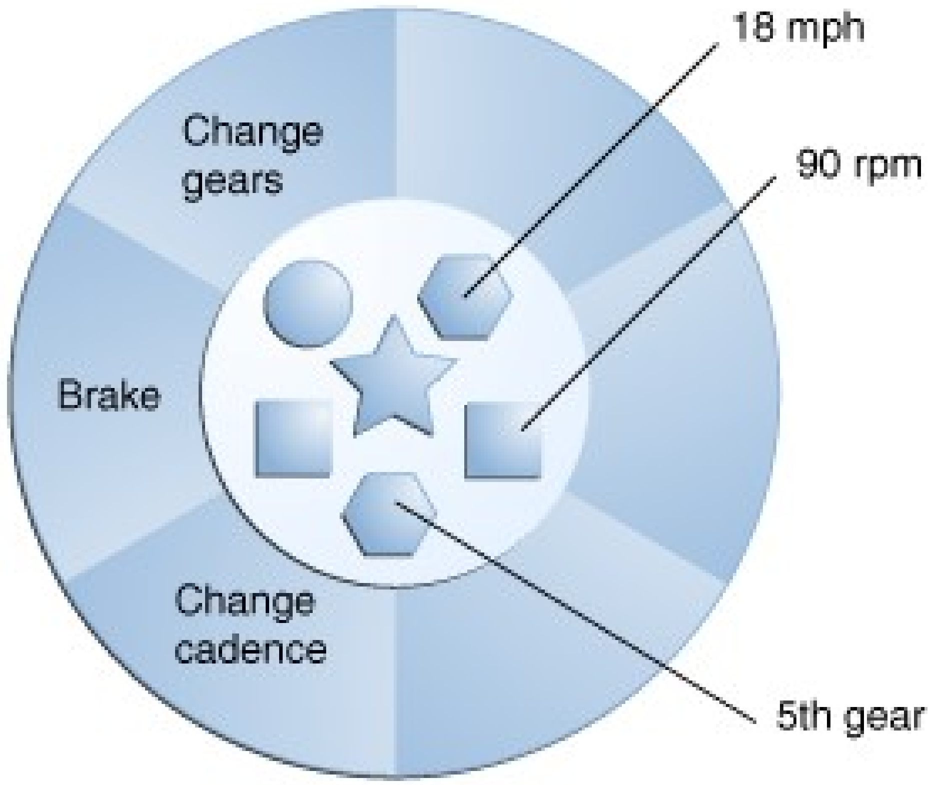
Object?

- 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.

Object?

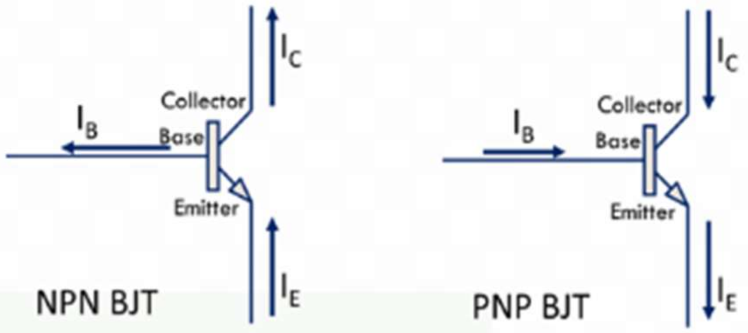


A software object.



A bicycle modeled as a software object.

Top 10 BJT Formulas



Transistor currents: $I_E = I_C + I_B$

DC current gain: $\beta_{DC} = I_C/I_B$

Base-to-emitter voltage (silicon): $V_{BE} \cong 0.7\ V$

Base current: $I_B = (V_{BB} - V_{BE})/R_B$

Collector-to-emitter voltage (common-emitter): $V_{CE} = V_{CC} - I_C R_C$

Collector-to-base voltage: $V_{CB} = V_{CE} - V_{BE}$

Approximate ac voltage gain: $A_V \cong R_C/r'_e$

Cutoff condition: $V_{CE(Cuttoff)} = V_{CC}$

Collector saturation current: $I_{C(sat)} = (V_{CC} - V_{CE})/R_C$

Minimum base current for saturation: $I_{B(min)} = I_{C(sat)}/\ \beta_{DC}$

Class?

- A *class* is the blueprint from which individual objects are created.
- It is a logical entity.

Class by Example

- In the real world, you'll 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 your bicycle is an *instance* of the *class of objects* known as bicycles

Class by Example

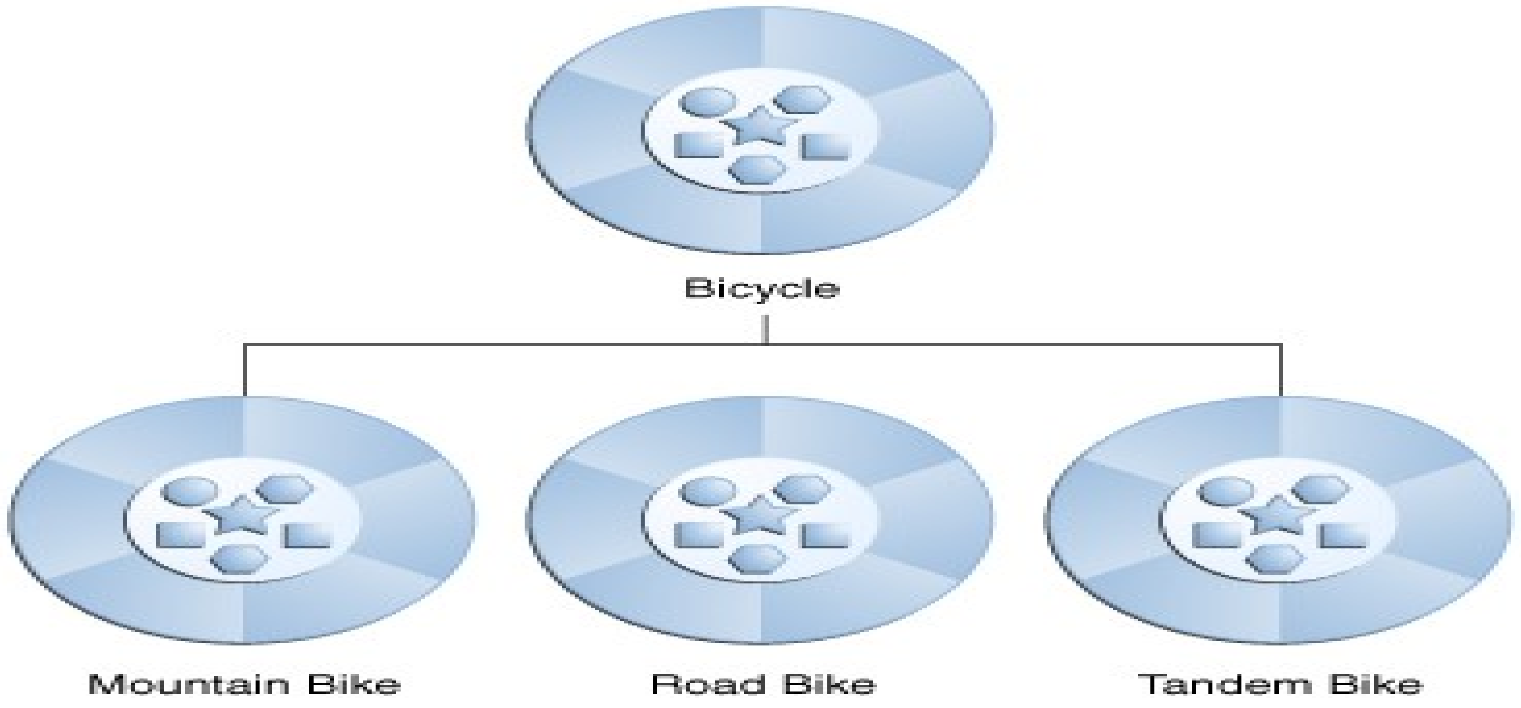
```
class Bicycle
{
int cadence = 0;
int speed = 0;
 int gear = 1;
void changeCadence(int newValue) {
cadence = newValue;
}
void changeGear(int newValue) {
 gear = newValue;
}
void speedUp(int increment) {
speed = speed + increment;
 }
void applyBrakes(int decrement) {
speed = speed - decrement;
}
void printStates() {
System.out.println("cadence:" +
cadence + " speed:" + speed + "
gear:" + gear);
}
}
```

Inheritance?

- **When one object acquires all the properties and behaviors of parent object** i.e. known as inheritance.
- It provides code reusability. It is used to achieve runtime polymorphism.
- Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.

Inheritance?

In this example, Bicycle now becomes the *superclass* of MountainBike, RoadBike, and TandemBike. In the Java programming language, each class is allowed to have one direct superclass, and each superclass has the potential for an unlimited number of *subclasses*:



Inheriting Syntax

- The syntax for creating a subclass is simple.
- At the beginning of your class declaration, use the extends keyword, followed by the name of the class to inherit from:

```
class MountainBike extends Bicycle {
// new fields and methods defining
// a mountain bike would go here
}
```

Interface?

- An interface is a group of related methods with empty bodies.
- Methods form the object's *interface* with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.
- Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler.

Interface?

- The interface in java is **a mechanism to achieve fully abstraction**.
- There can be only abstract methods in the java interface not method body.
- It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.

Package?

- A package is a namespace that organizes a set of related classes and interfaces.
- Conceptually you can think of packages as being similar to different folders on your computer.
- The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the "Application Programming Interface", or "API" for short.
- Its packages represent the tasks most commonly associated with general-purpose programming.

Polymorphism?

- When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.
- In java, we use method overloading and method overriding to achieve polymorphism.
- Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

"Now Speak!"
"Quack"
"Woof"
"Meow"

Abstraction?

- **Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.
- In java, we use abstract class and interface to achieve abstraction.

Encapsulation?

- **Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Thank you