SKILLSTORM

# OOP

Object real life representation.

## Encapsulation

Take everything about a thing and put it in one place

## Inheritance

A way of reusing classes. Not repeating code

## Abstraction

Create a class that generalize common behavior. Clocks having hour and minute

## Polymorphism

Many forms. Change behavior under different circumstances

### Classes

Blueprints used to define objects

* Real world objects
* Application components (code)

Includes:

* Variables (state)
* Methods (behavior)
* Constructors (initialize the state); no return type; used to initialize the state of variables. i.e., speed = 0 for Vehicle class.
* Parameters – value passed through method
* class Vehicle {// class body
* int speed; // declared variable
* void accelerate() {//method body
* }
* Vehicle(){
* speed = 0;
* }//Constructor
* }
* public class HelloWorld {
* public static void main(String[] args) {
* System.out.println("Hello World");
* }
* }

# Classes and Objects

### Data Types

* Primitive: numeric, text, true/false
* Non-primitive: Any class wanting to be used; create your own
* class Vehicle {// class body
* //Primitive
* int speed = 2000000; // declared variable
* byte wheels = 4;
* short weight = 2000;
* char fuel = 'D';
* boolean running = false;
* long serial = 123456789123456789L;
* float fuelRemaining = 100.1234567890123f;
* String make = "Toyota";
* //Can use own class as data type
* Vehicle toy;
* void accelerate() {//method body
* int count = 0;
* }
* }

### Arrays

* Stores multiple values in a single reference of the same data type
* Sequential block of memory
* Must provide dimensions to declare
* public class Arrays {
* public static void main(String[] args) {
* int[] array = new int[6];
* array[0] = 155;
* array[5] = 77;
* String[] words = {"abc", "def", "jkl"};
* for (int i = 0; i < words.length; i++) {
* System.out.println(words[i] + "");
* }
* }
* }

## Method Declaration and Invocation

* Declaration: create a method: Public void go (int x, String y) {}
* Invocation: call a method: object.go(10,”A”); Public void go (int x) {} object.go(100);

-Method overloading

Static methods are called without an object. E.g. Math.random();

public class Method {

    Method example = new Method();

    int returned = example.go(); //non-static must use object

    int value = Method.go(5); //static uses class

    public int go(){

        return 11;

    }

    public static int go(int x){

        return x;

    }

}

## Constructors

* Used to initialize an object’s state
* No argument constructor: ex:
* Pizza pizza = new Pizza(); = blank pizza Pizza pizza = new Pizza(“Pepperoni”); = pepperoni pizza

- constructor overloading

\*Objects are instances of classes.\*

public class Constructors {

    public static void main(String[] args) {

        Pizza pizza = new Pizza();

        pizza.slices = 8;

        Pizza pie = new Pizza(6);

        pie.topping = "pepperoni";

    }

}

class Pizza {

    int slices;

    String topping;

    Pizza() {} // must define no argument constructor to create instance of pizza

    Pizza(int pieces){

        slices = pieces;

    }

    public void eat(){ //each overloaded constructor will have access to this method

        slices -= 1;

    }

}

# Java Virtual Machine

## Compilation and Bytecode