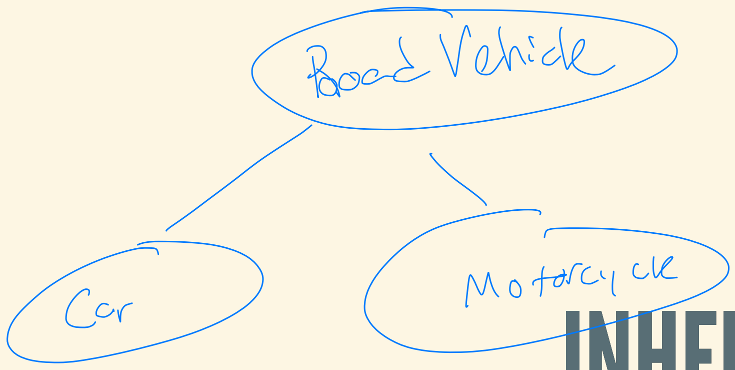
INHERITANCE

MOTIVATION

- Physically, items often share common aspects
- Example:
    - cars, semis, motorcycles are all road vehicles
    - all have wheels (different #s)
    - some have windows, ...

MOTIVATION

- Coding each separately would repeat any common aspects
- Logically different -> makes sense to code as different things
- How can we represent and capitalize on logical relationship?



INHERITANCE - BASIC IDEA

- Represent connections between classes via inheritance
- parent/super/base class
- child/sub class
- allows for code/software reuse

IS-A VS HAS-A

- "has-a" relationship represents variables in classes
- "is-a" relationship signals potential child
- Example:
    - mustang is a car
    - car is a road vehicle
    - car has windows, trunk
- Example:
    - horse is a mammal
    - dog is a mammal
- If X is derived from Y, should be able to say X is a Y

IMPLEMENTATION

- In Java, denoted with `extends` keyword

```
public class Car extends Vehicle
{
    //code here
}
```

only say a child class

WHAT HAPPENS

- child inherits methods and variables from parent
- parent gets nothing from child
- private methods/variables cannot be referenced
    - still exist
- constructors not inherited

```
public class Vehicle {
    public int nwheels;

    public void honk() {
        System.out.println("beep")
    }
    public Vehicle ( ) {
        this.nwheels = 4
    }
}

public class Car extends Vehicle {
    public int nwindows;
    public Car (int nwindows) {
        super ( );
        this.nwindows = nwindows;
    }
}
```

```
~~public class Car {~~
    ~~public Car(int nwindows) {~~

    ~~}~~
    ~~public Car(int nwheels {~~

    ~~}~~
}

            Car c = new Car(4)
```

this.myvar

CONSTRUCTORS

- What if you need/want to use parent constructor?
- `super` reference -> references parent
- could just set same variables, but better practice to let parent class handle
- call to super should be first line
- no call -> automatically calls parent with no parameters
- no explicit parent -> implicit is `Object`

SINGLE VS MULTIPLE

- Java only allows single inheritance
- One class can't inherit from 2 parents (multiple inheritance)
- But, multiple classes can inherit from same parent (siblings)

OVERRIDING METHODS

- Defining method with same name overrides parent
- Very common
- Examples:
    - `toString`
    - `equals`

equals (Object o)

- Not the same as method overloading

FINAL KEYWORD

- `final` methods cannot be overridden
- `final` classes cannot be inherited from

ABSTRACT CLASSES

- Sometimes parent classes are for organization and structure
- May want to represent higher level thing that isn't actually an object we want to instantiate
- Example 1:
  - Parent class = Animal
  - Child classes = Dog, Cat, Lion
- Example 2:
  - Parent class = Vehicle
  - Child class = Car, Semi, Motorcycle

ABSTRACT CLASSES

- Solution: abstract class
- Cannot be instantiated
- Use `abstract` keyword

```
abstract class Animal {
}
```

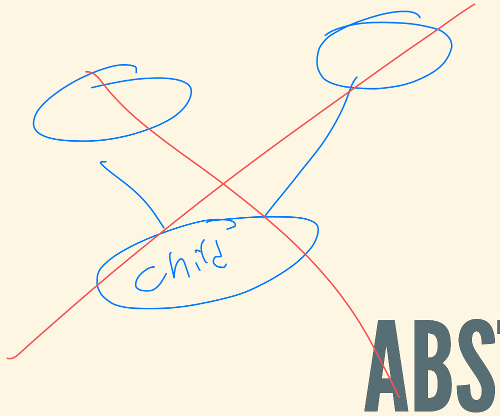
ABSTRACT METHODS

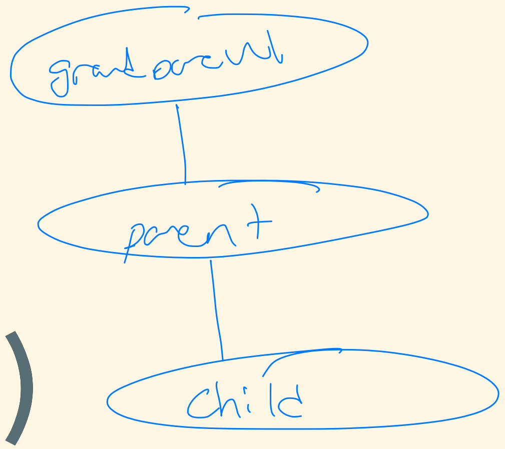
- Method without implementation (body)
- Used in abstract classes
- Forces child classes to implement
- Declare with a method prototype

```
public abstract void makeNoise();
public abstract class Vehicle {
    public abstract honk();


    public void fuelUp() {
        // do stuff
}   }

public class Car extends Vehicle {
    public honk() {
        // do stuff
    }
}
```





ABSTRACT CLASSES (CONT.)

- If child does not implement all `abstract` methods -> child must be abstract
- Have no use until extended by another class
- Can have concrete methods as well
- Can still have constructors

protected

alternative public/private