

مفاهیم شی گرایى در جاوا

# شی گرایى چیست؟

ساختارى براى كدنويسى

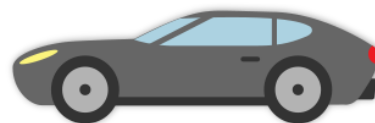
سازماندهى كدها

برگرفته شده از دنيای واقعى

# مفاهیم شی گرای

- شی ( Object )
- کلاس ( Class )
- رفتار ( Behavior )
- صفت ( Attribute )
- چند ریختی ( Polymorphism )
- کپسوله سازی ( Encapsulation )
- ارث بری ( Inheritance )
- انتزاع یا تجرید ( Abstraction )

# مثال : خودرو ها!



# مثال



# اشياء ( Objects )

پرادو



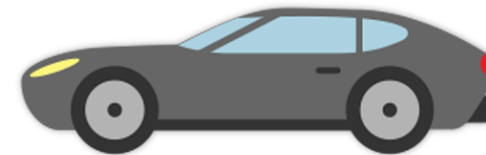
پورشه



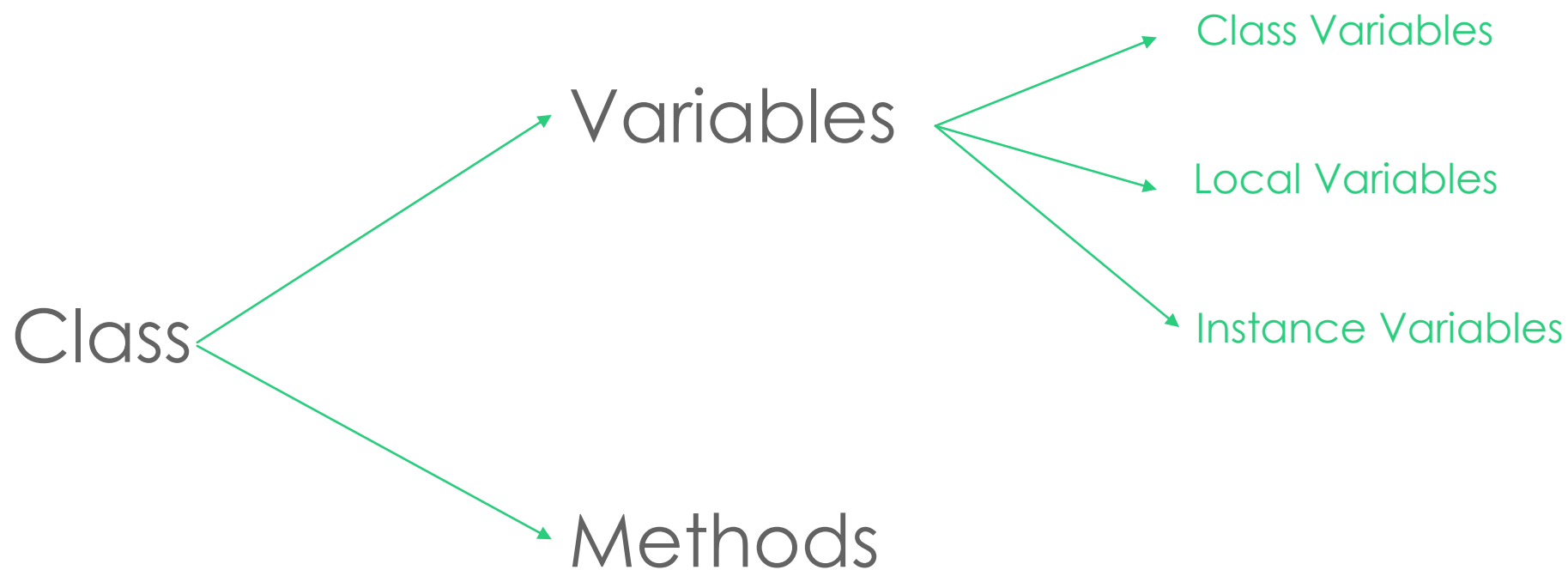
پراید هاچ بک



مازراتی



# اجزای تشکیل دهنده کلاس



# انواع متغیرها

```
public class Car {
```

```
    //Class Variable
```

```
    private static int status =1;
```

```
    //Instance Variable
```

```
    private int currentSpeed=20;
```

```
    public Car(){
```

```
    }
```

```
    private void drive(){
```

```
        //Local Variable
```

```
        boolean isDriving=false;
```

```
    }
```

```
    private void brake(){
```

```
    }
```

```
}
```

Class Variables

Instance Variables

Local Variables



# متد سازنده ( Constructor )

```
public class Car {  
  
    //Class Variable  
    private static int status =1;  
  
    //Instance Variable  
    private int currentSpeed=20;  
  
    public Car() { ← Constructor  
  
    }  
    private void drive() {  
  
        //Local Variable  
        boolean isDriving=false;  
  
    }  
    private void brake() {  
  
    }  
  
}
```

# ساختن یک Object

سه مرحله برای ساختن یک شی:

1. تعریف متغیر برای شی

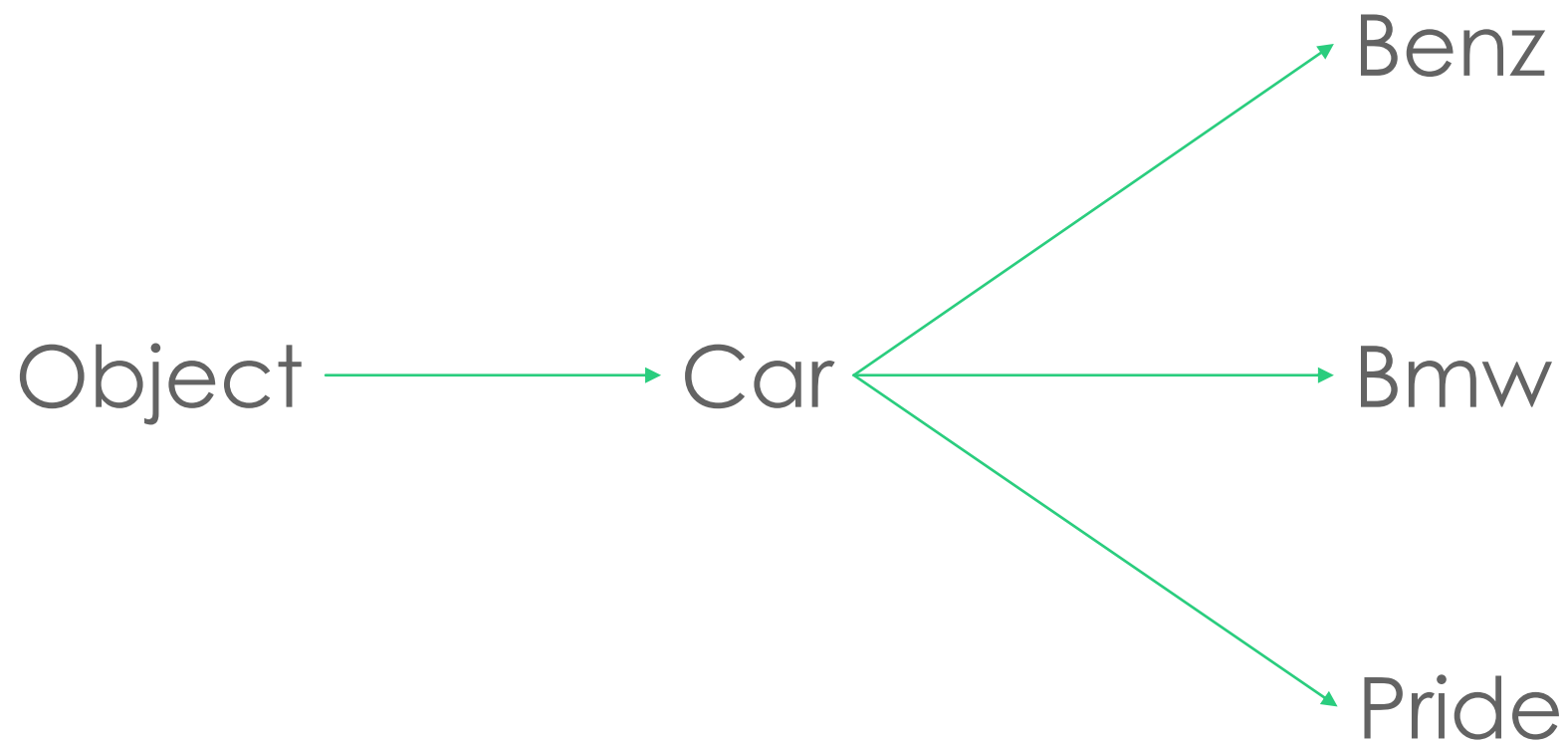
2. استفاده از کلمه ی new برای ساخت شی جدید

3. فراخواندن متد سازنده ( Constructor ) بعد از کلمه ی new


1      2      3

```
Car car=new Car ("Benz" );
```

# ارث بری ( Inheritance )



# ارث بری ( Inheritance )



The diagram illustrates the inheritance relationship between two classes. A green arrow points from the word 'Benz' in the code to the label 'Subclass'. Another green arrow points from the word 'Car' in the code to the label 'Superclass'.

```
public class Benz extends Car {  
  
    private int gpsStatus=0;  
  
    public Benz () {  
  
    }  
  
}
```

# Overriding

پیاده سازی یک رفتار به ارث برده از Super Class و تغییر آن بر اساس نیازهای خود است.

# مثال

```
public class Benz extends Car {  
  
    private int gpsStatus=0;  
    public Benz () {  
  
    }  
  
    @Override  
    public void drive() {  
        super.drive();  
    }  
  
    @Override  
    public void brake() {  
        super.brake();  
    }  
}
```

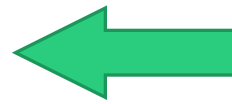
# قوانین مهم Overriding

1. لیست آرگومان های تابع دقیقا، همان تعداد آرگومان های تابع Override شده باشد.
2. Return type تابع دقیقا همان Return type تابع Override شده باشد.
3. تابعی که Final تعریف شده باشد، قابل Override شدن نیست.
4. تابعی که Static تعریف شده باشد، قابل Override شدن نیست ولی قابل دوباره تعریف کردن است.
5. Constructor ها قابل Override شدن نیستن.

# Overloading

```
public Car() {  
}  
public Car(String name) {  
}  
public Car(int id, String name) {  
}
```

مثال



اگر یک تابع چندین بار با یک نام

ولی با آرگومان های مختلف پیاده سازی شود

Overloading نام دارد.



# کپسوله سازی ( Encapsulation )

جدا سازی Variable های یک کلاس از دید سایر کلاس ها ( Data Hiding )

موارد مورد نیاز برای دستیابی به Encapsulation:

1. Private تعریف کردن متغیر های کلاس

2. نوشتن Getter و Setter برای دسترسی به Variable ها

# کپسوله سازی ( Encapsulation )

```
public class EncapuslationExample {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private String email;  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
}
```

# مزایای Encapsulation

1. قابلیت Read only یا Write only کردن متغیر های کلاس.
2. کنترل کامل داشتن کلاس روی اینکه چه دیتایی داخل متغیرهایش ذخیره شود.
3. شخصی که از کلاس استفاده می کند، اصلا نمی داند چطور دیتا داخل کلاس ذخیره می شود. مثلا اگر نویسنده کلاس تصمیم بگیرد نوع متغیر را عوض کند، کسی که از کلاس او استفاده می کرده، نیازی به تغییر کد ندارد.

# انتزاع یا تجرید ( Abstraction )

پیاده سازی نکردن جزییات و فقط مشخص کردن عملکرد های کلاس را Abstraction می گویند.

چه زمانی از abstraction استفاده می کنیم؟

**جواب :** زمانی که می دانیم چه کارهایی باید انجام شود، اما نمی دانیم چطوری باید انجام شود!

```
public abstract class AbstractCar {  
    private String name;  
  
    abstract void driving();  
    abstract void stop();  
  
    private String getName() {  
        return name;  
    }  
  
    private void setName(String name) {  
        this.name=name;  
    }  
}
```

# چطور از کلاس Abstract نمونه سازی کنیم؟

```
public class Bmw extends AbstractCar {  
    @Override  
    void driving() {  
  
    }  
  
    @Override  
    void stop() {  
  
    }  
}
```

```
AbstractCar abstractCar=new AbstractCar();
```

```
Bmw bmw=new Bmw ();
```

# Interface

Interface شبیه کلاسی است که فقط شامل Abstract Method باشد.



```
public interface InterfaceExample {  
    abstract void doThis();  
    abstract void doThat();  
}
```

# چگونگی استفاده از Interface

~~`public class` EncapsulationExample `extends` InterfaceExample~~

`public class` EncapsulationExample `implements` InterfaceExample



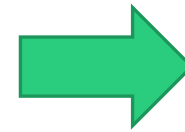
# چند ریختی Polymorphism

قابلیتی که یک شی می تواند اشکال مختلفی به خود بگیرد.

```
public class Vehicle extends Object
```

```
public class Car extends Vehicle
```

```
public class Benz extends Car
```



## Polymorphism

```
Benz benz=new Benz();  
Car car=benz;  
Vehicle vehicle=benz;  
Object object=benz;
```

- <https://docs.oracle.com/javase/tutorial/java/concepts>
- <http://www.javatpoint.com/java-oops-concepts>
- <http://www.tutorialspoint.com/java>