

Java Classes

Writing our own classes to model objects



1

Java Classes



- A class of objects of the same type are modeled using a Java class.
- A Java class contains:
 - fields - store the "state" of an object
 - methods - represent the "behaviors" that this object can perform
 - some methods may create an object of this class (constructors)
 - some methods may change the state of the object (mutators)
 - some methods may access information about the current state of the object (accessors)

2

Object Fields



- Every object of a class has a set of properties that define the object
 - Example: If an object belongs to the Car class, two of its properties are the make and the mileage.
- Each property has some "value" once the object is created
 - Example: make = "Olds", mileage = 15110
- The values of the properties defines the state of the object

3

Fields

(sometimes called *instance variables*)

in Car class



```
public class Car
{
    private String make;
    private int mileage;

    // methods go here
}
```

All object fields should be defined as **private**.

4

Constructors



- A constructor is a method that creates an instance of the class.
 - An instance is one member of the class.
 - Example:
In a program we might write
`Car myCar = new Car("Olds", 15110);`
This would cause the constructor of the **Car** class to execute, giving it two arguments to define the initial state of the car.



5

Constructors

in Car class



```
public Car(String carMake,
            int initialMileage)
{
    make = carMake;
    mileage = initialMileage;
}
```

Don't use the same names for the parameter variables and the object fields.

6

A sample program that uses the Car class

```
public class CarManager {
    public static void main(String[] args) {
        Car myCar = new Car("Olds", 15110);
        // instructions to use this car
        // goes here...
    }
}
```

in
CarManager
class

7

```
public class Car {
    private String make;
    private int mileage;

    public Car(String carMake, int initialMileage) {
        make = carMake;
        mileage = initialMileage;
    }
    // other methods go here
}

public class CarManager {
    public static void main(String[] args) {
        Car myCar = new Car("Olds", 15110);
        // instructions to use this car goes here
    }
}
```

PARAMETERS

carMake [] → "Olds"
initialMileage [15110]

myCar [] → make [] → "Olds"
mileage [15110]

8

Constructors

- We may have more than one constructor.
 - Another constructor might be used if we don't know the complete state of an object.
 - We could use default values for properties that are not specified.
 - Example:
In our main method, we might write
`Car myOldCar = new Car("Saturn");`

in
CarManager
class



9

Constructors

```
public Car(String carMake)
{
    make = carMake;
    mileage = 0;
}
```

We use a default value of 0 for the mileage if it is not specified.

in
Car
class

This is another example of overloading.

10

Accessors

- An accessor is a method that accesses the object without changing its state.
 - Example:
In our main method, we might write
`int totalMiles = myCar.getMileage() + myOldCar.getMileage();`

in
CarManager
class

This would cause the `getMileage` method to be executed, once for each car. This method does not require any arguments to be sent in from our program in order to do its job.

11

Accessors

```
public int getMileage()
{
    return mileage;
}
```

All methods except the constructor require a return type (the type of the result that the method returns back once it's done).

in
Car
class

12

Accessors

in
Car
class

return type

```
public double getCostOfOwnership()  
{  
    double cost = 0.45 * mileage;  
    return cost;  
}
```

This accessor calculates and returns the cost of ownership for a car, assuming it costs 45 cents to maintain the car per mile driven. Note that the state of the car does not change when this method runs.

13

Accessors

in
CarManager
class

- How would we call the `getCostOfOwnership` from the main method of `MyProgram`?

```
Car.getCostOfOwnership();           NO  
myCar.getCostOfOwnership();        NO*  
myCar.getCostOfOwnership(15100);   NO  
System.out.println(  
    myCar.getCostOfOwnership());    YES  
double myCost =  
    myCar.getCostOfOwnership();     YES
```

14

Mutators

- A mutator is a method that could change the state of an object in this class.

- Example:

In our main method, we might write

```
myCar.drive(500);    // drive 500 miles
```

This would cause the `drive` method to be executed, with the integer argument 500 sent to this method to indicate how many miles the car was driven.

15

Mutators

in
Car
class

return type

```
public void drive(int miles)  
{  
    mileage = mileage + miles;  
}
```

Since this method performs a computation but does not return a result, the return type is specified as `void`.

16

Mutators

in
CarManager
class

- If we call a void method, then this call is written as single instruction in a program.
- It is not embedded in other operations.

```
myCar.drive(500);           OK  
System.out.println(  
    myCar.drive(50));       NO  
int totalMiles =  
    myCar.getMileage() +  
    myCar.drive(50);       NO
```

17

Mutators

in
Car
class

```
public void resetMileage()  
{  
    // this is normally illegal  
    // in real life  
    mileage = 0;  
}
```

To call this method from `MyProgram`:
`myOldCar.resetMileage();`

18

toString

in
Car
class



- A special accessor named **toString** is a method that returns a string containing the current state of the object.

```
public String toString() {  
    return "Make = " + make +  
        ", Mileage = " + mileage;  
}
```

- Note: The signature MUST be as shown.
- This method is typically used for debugging.

19

Using toString

in
CarManager
class



```
public class CarManager {  
    public static void main(String[] args) {  
        Car myCar = new Car("Olds", 15110);  
        Car myOldCar = new Car("Saturn");  
        myCar.drive(1234);  
        myOldCar.resetMileage();  
        System.out.println(myCar);  
        System.out.println(myOldCar);  
    }  
}
```

When you try to print an entire object (as shown), you automatically call its **toString** method.

20

Writing our own equals method



- Example: Two cars are equal if they have the same make and the same mileage.

```
public boolean equals(Car otherCar)  
{  
    if (this.make.equals(otherCar.make)  
        && this.mileage == otherCar.mileage)  
        return true;  
    else  
        return false;  
}
```

this refers to the object
running this method

- Usage: `if (myCar.equals(myOldCar)) ...`

21

Writing our own equals method

(cont'd)



```
public boolean equals(Car otherCar)  
{  
    // alternate version  
    return (make.equals(otherCar.make)  
        && mileage == otherCar.mileage);  
}
```

More about **toString** and **equals** later in the semester...

22

Revisiting the Die class



```
public class Die {  
    private int faceValue;  
    public Die() {  
        faceValue = 1;  
    }  
    public void roll() {  
        faceValue = (int)(Math.random()*6)+1;  
    }  
    public int getFaceValue() {  
        return faceValue;  
    }  
}
```



23

Revisiting the Die class

(cont'd)



```
public _____ toString() {  
    _____;  
}  
public _____ equals(Die otherDie) {  
    _____;  
}  
// end of Die class
```

24