CS102
Programming II

CLASS (PART - I)

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN
UNIVERSITY

TOPICS

- Classes & objects
- Constructors
- **`this`** keyword
- Encapsulation
- **`static`** attributes and methods
- **`final`** attributes
- **`toString()`** method
- Object Composition
- **`equals()`** and **`instanceof`**

CS102 Programming II

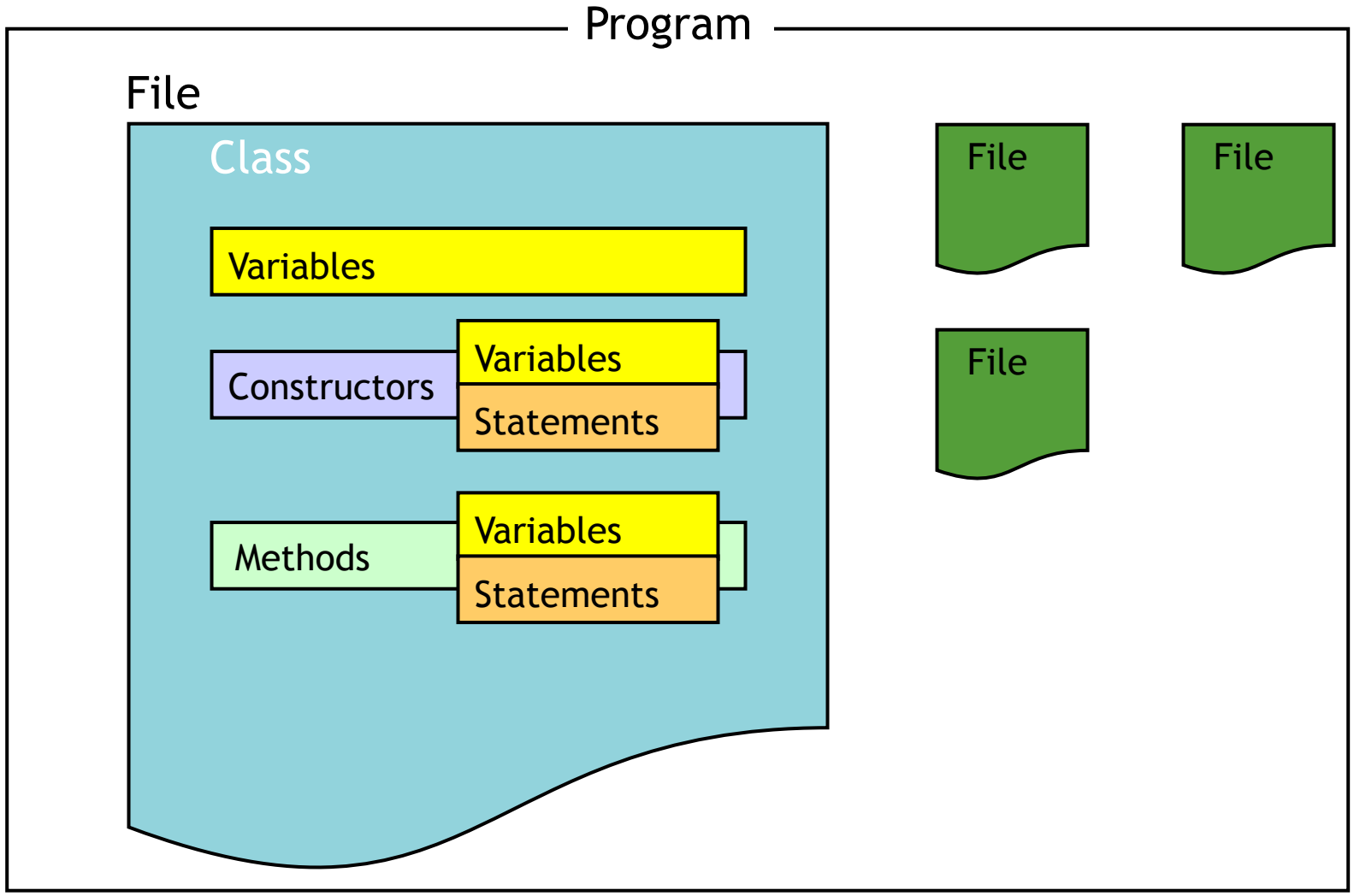
CLASSES AND OBJECTS

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الامير سلطان
PRINCE SULTAN UNIVERSITY

DIAGRAM OF PROGRAM STRUCTURE

- A program consists of one or more classes
- Typically, each class is in a separate .java file

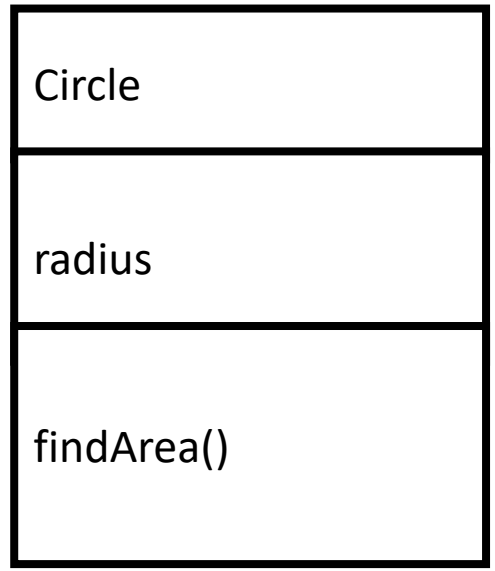


CLASSES

- A *class* is a collection of *attributes* (data) and *methods*

```
public class Circle {
   double radius = 1.0; //attribute

   double findArea(){ // method
     return radius * radius * 3.14159;
   }
}
```



CREATING OBJECTS

An object is an instance of a class.

```
ClassName objectReference = new ClassName();
```

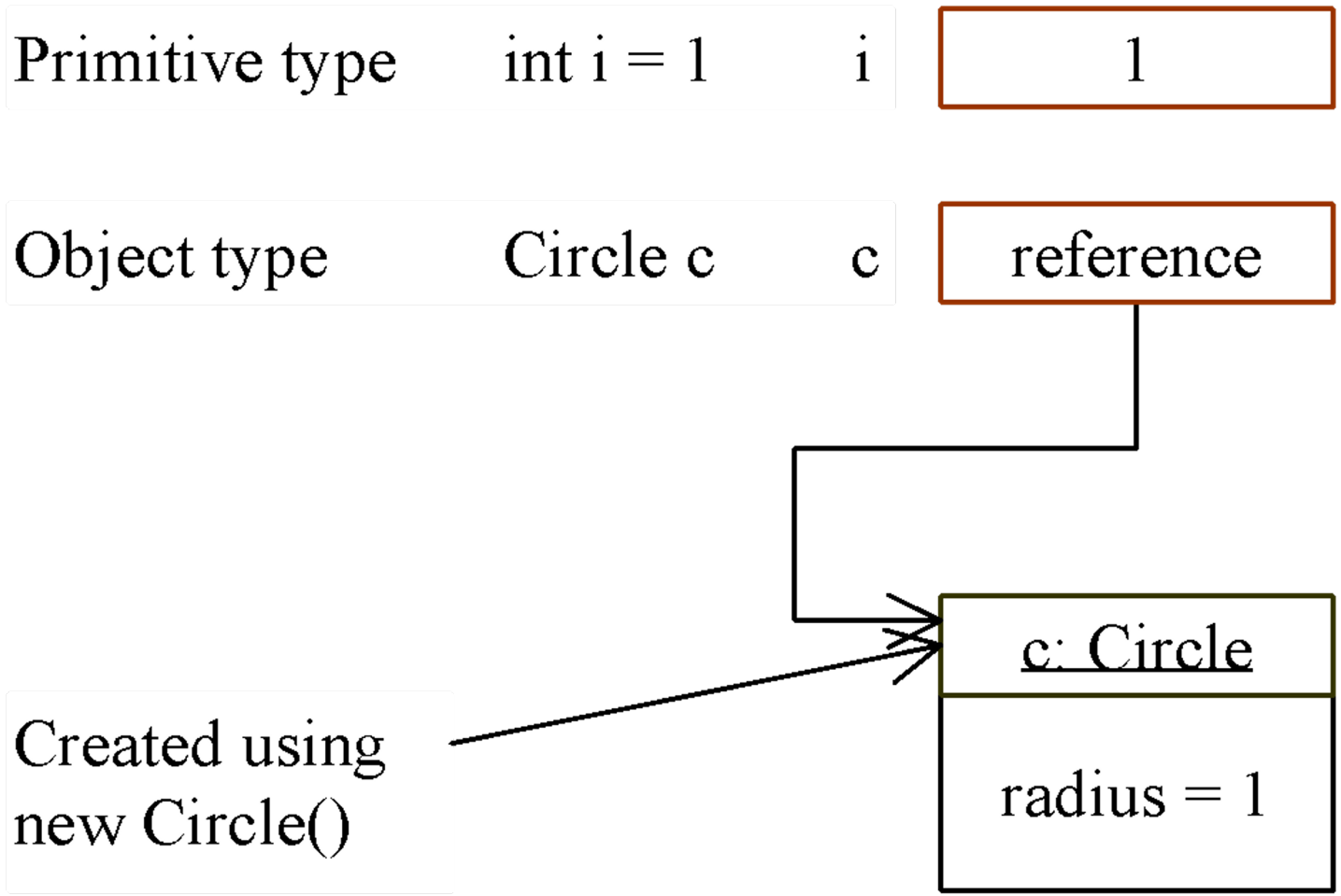
Example:

```
Circle myCircle = new Circle();
```

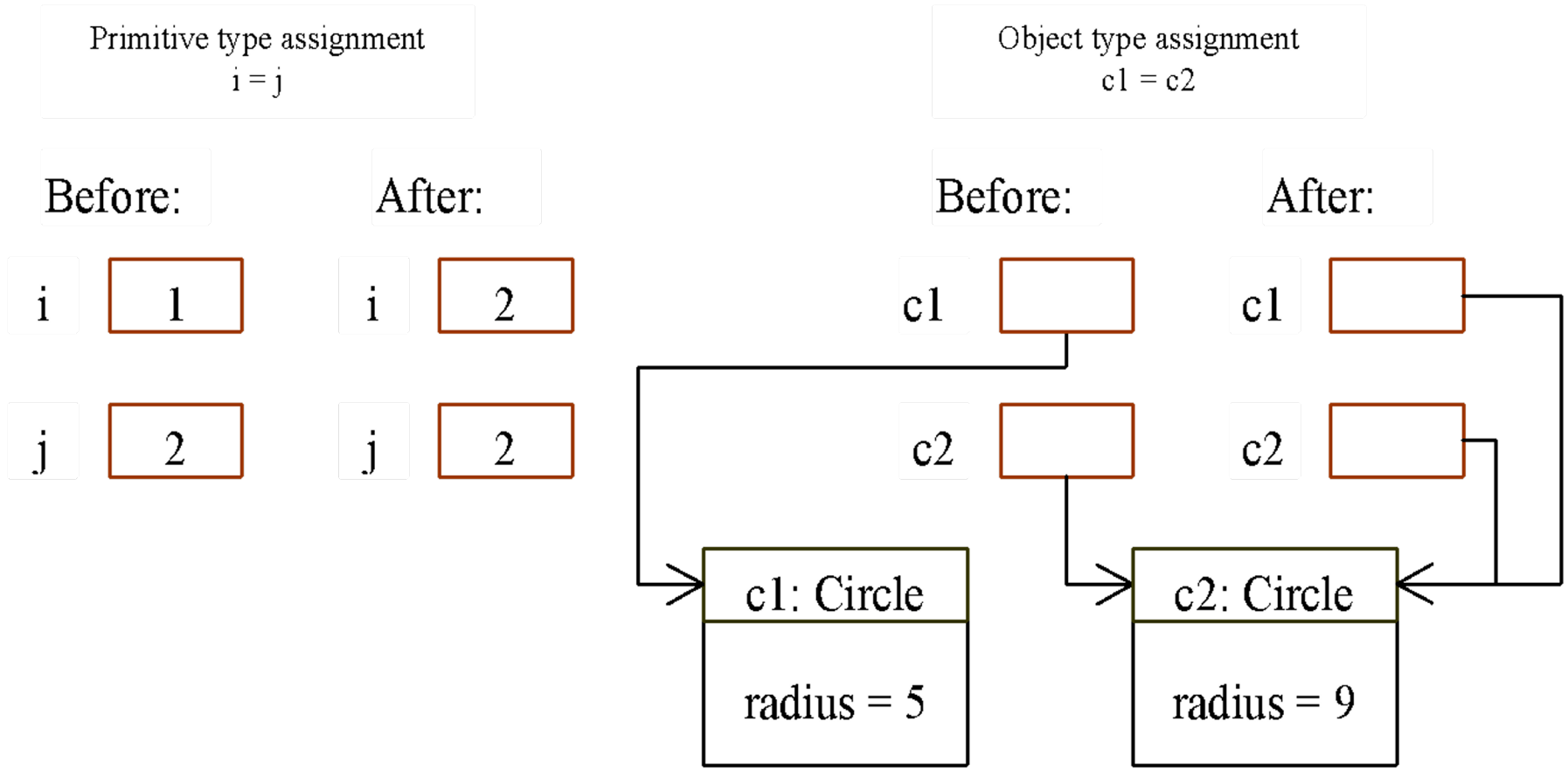
The object reference is assigned to the object reference variable.

`myCircle` is a object of class **`Circle`**

PRIMITIVE DATA TYPE VS OBJECT TYPES



PRIMITIVE DATA TYPE VS OBJECT TYPES



ACCESSING OBJECTS

- Referencing the object's data:

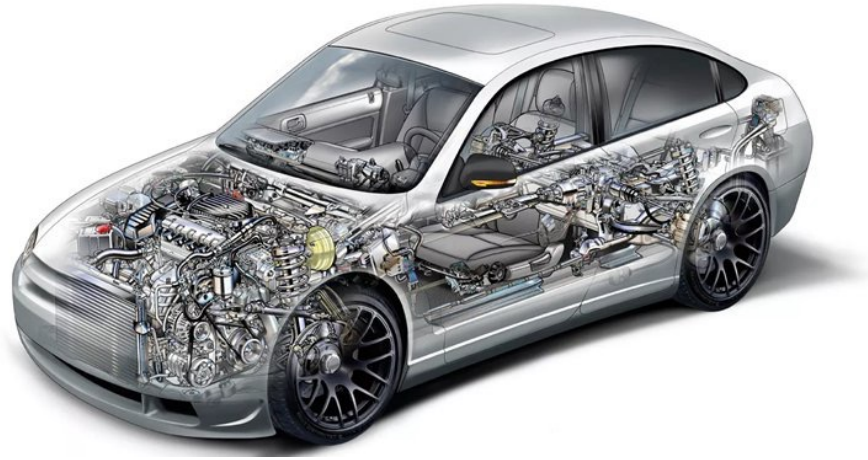
```
objectReference.data
myCircle.radius
```

- Invoking the object's method:

```
objectReference.method
myCircle.findArea()
```

EXERCISE

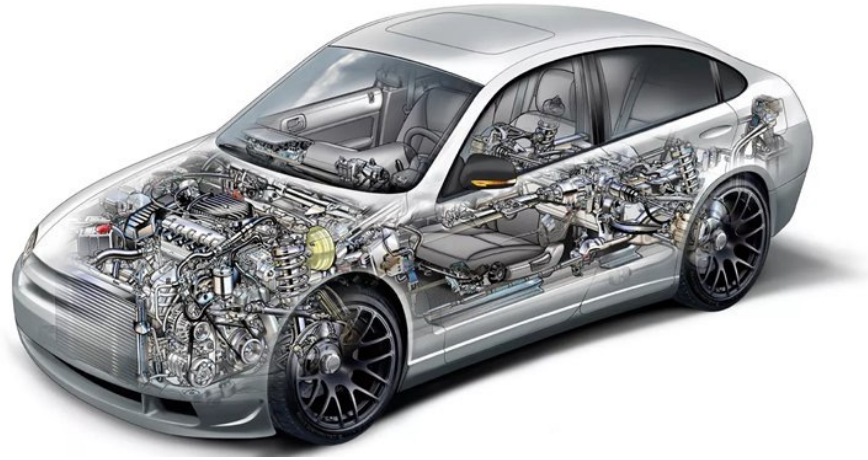
- Make a class called **`Car`**.
- It has two attributes, **`year`** and **`model`**.
- It has one method **`getYear`** that returns **`year`**.



EXERCISE/SOLUTION

```
public class Car {
   int year;
   String model;

   public int getYear(){
      return year;
   }
}
```



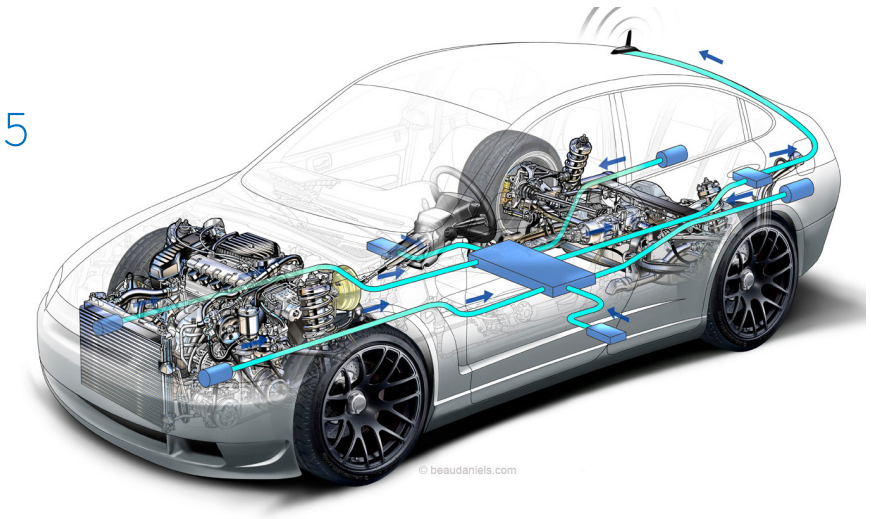
EXERCISE

- Make a tester program.
- Make an object **`A`** of type Car. Set **`year`** and **`model`** for the object.
- Get the value of **`year`** from object **`A`** and print it.

EXERCISE

- Make a tester program.
- Make an object **A** of type Car. Set **year** and **model** for the object.
- Get the value of **year** from object **A** and print it.

```
public static void main(String [] args){
   Car A = new Car();
   A.year = 2025;
   A.model = "Lexus 400";
   System.out.print(A.getYear());   //prints 2025
}
```



CS102 Programming II

CONSTRUCTORS

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

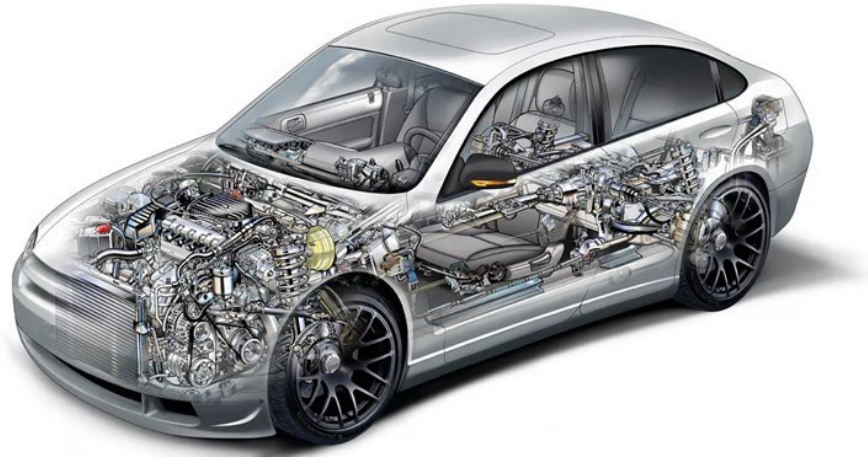
جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

WHAT IS A CONSTRUCTOR?

- Constructor is a special method that gets invoked “automatically” at the time of object creation.
- Constructor is used for initializing objects with default values.
- Constructor has the same name as the class name.
- Constructor cannot return values.
- A class can have more than one constructor as long as they have different signature.

CONSTRUCTOR

```
public class Car {
   int year;
   String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   public int getYear(){
      return year;
   }
}
```



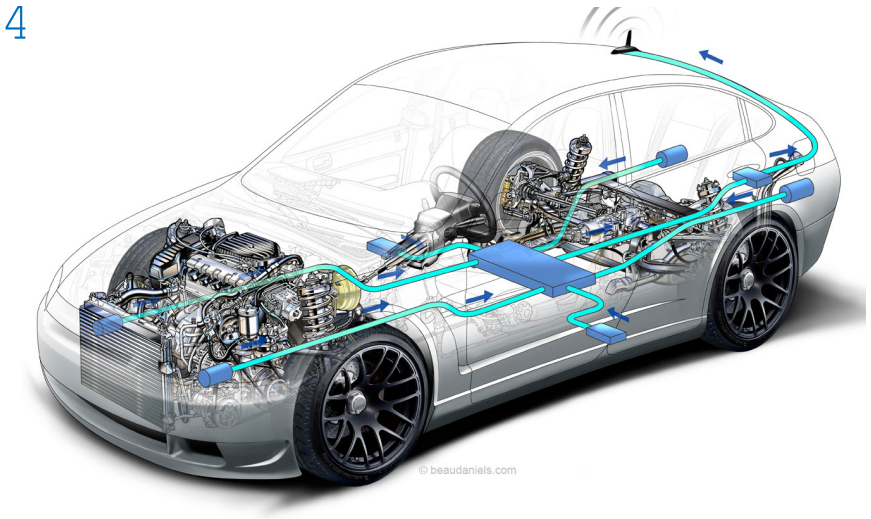
EXERCISE

- Modify the tester program. Make a object **B** of type **Car**. Use
- Get the value of **year** from object **B** and print it.

```
public static void main(String [] args){
   Car B = new Car();
   B.year = 2025;
   B.model = "Lexus 400";
   System.out.print(B.getYear()); // prints 2024

}
```

```
Car(){
    year = 2024;
    model = "Lexus";
  }
```



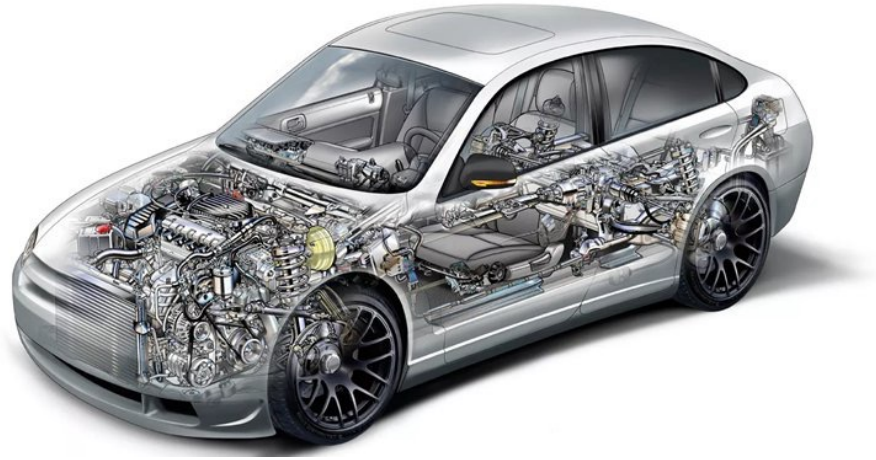
METHOD OVERLOADING

- Constructors all have the same name.
- Methods are distinguished by their signature:
  - name
  - number of arguments
  - type of arguments
  - position of arguments
- That means, a class can also have multiple usual methods with the same name.

CONSTRUCTOR OVERLOADING

```
public class Car {
   int year;
   String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   Car(int y, String S){
      year = y;
      model = S;
   }
   public int getYear(){
      return year;
   }
}
```

←Overload



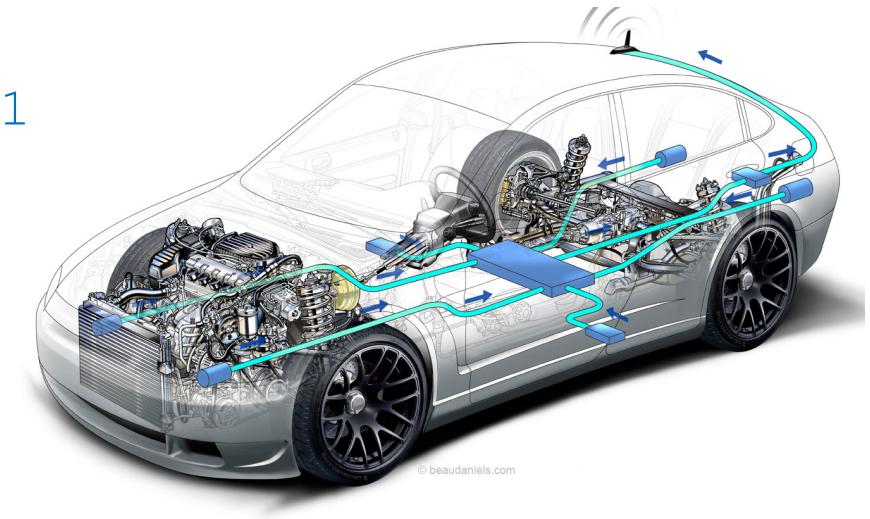
EXERCISE

- Modify the tester program. Make a object **C** of type **Car**.
- Get the value of **year** from object **C** and print it.

```
public static void main(String [] args){
   Car B = new Car();
   System.out.print(B.getYear()); // prints 2024

   Car C = new Car(2021, "Ford 150");
   System.out.print(C.getYear()); // prints 2021
}
```

```
Car(int y, String S){
   year = y;
   model = S;
}
```



CS102 Programming II

THIS

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

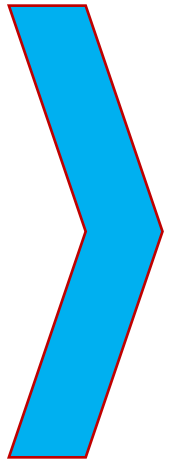
جامعة الامير سلطان
PRINCE SULTAN UNIVERSITY

`this` KEYWORD

- **this** keyword is used to refer to the class itself
- It must be used to **differentiate attributes names from parameter names** when they have the same names

this KEYWORD

```
public class Car {
   int year;
   String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   Car(int year, String model){
      year = year;
      model = model;
   }
   public int getYear(){
      return year;
   }
}
```



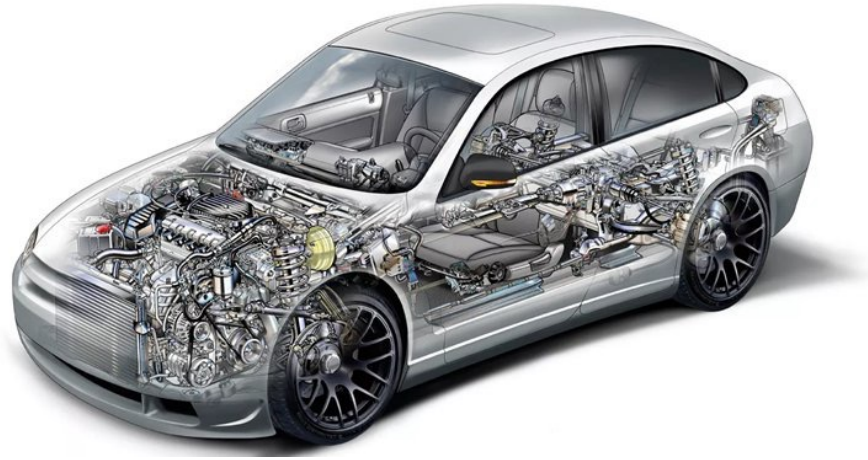
```
public class Car {
   int year;
   String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   Car(int year, String model){
      this.year = year;
      this.model = model;
   }
   public int getYear(){
      return year;
   }
}
```

COPY CONSTRUCTOR

- A **copy constructor** is a constructor that creates an object using another object of the same Java class.
- It helps when we want to make a **deep copy** of an existing object.
- **Deep copy:** copies all attributes from one object to another.

COPY CONSTRUCTOR

```
public class Car {
   int year;
   String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   Car(Car C){
      if(C!=null){
         this.year = C.year;
         this.model = C.model;
      }
   }
   public int getYear(){
      return year;
   }
}
```



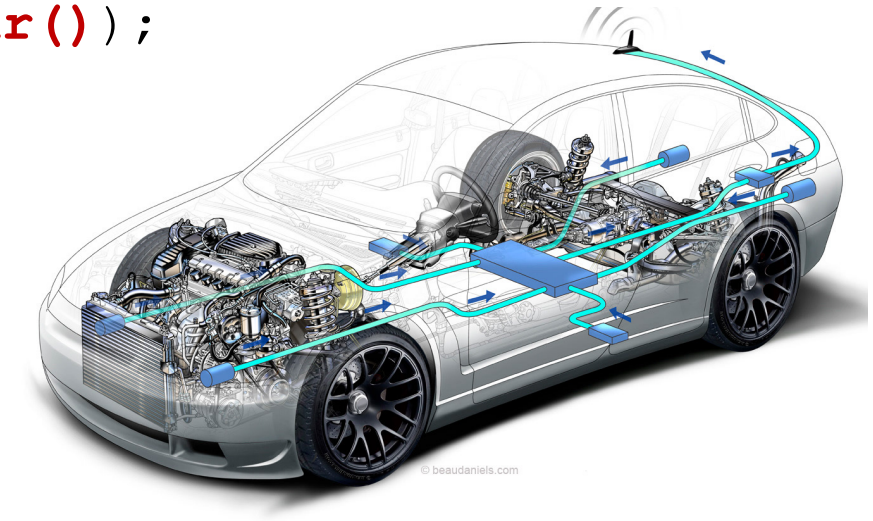
EXERCISE

- Make two objects **D1, D2** of type **Car**.
- Copy **D1** to **D2**.

```
Car(){
    year = 2024;
    model = "Lexus";
 }
```

```
Car(Car C){
    if(C!=null){
      this.year = C.year;
      this.model = C.model;
    }
 }
```

```
public static void main(String [] args){
   Car D1 = new Car();
   Car D2 = new Car(D1);
   System.out.println(D1.model +" "+ D1.getYear()); // prints Lexus 2024
   System.out.println(D2.model +" " + D2.getYear());
   // also prints Lexus 2024
}
```



CS102 Programming II

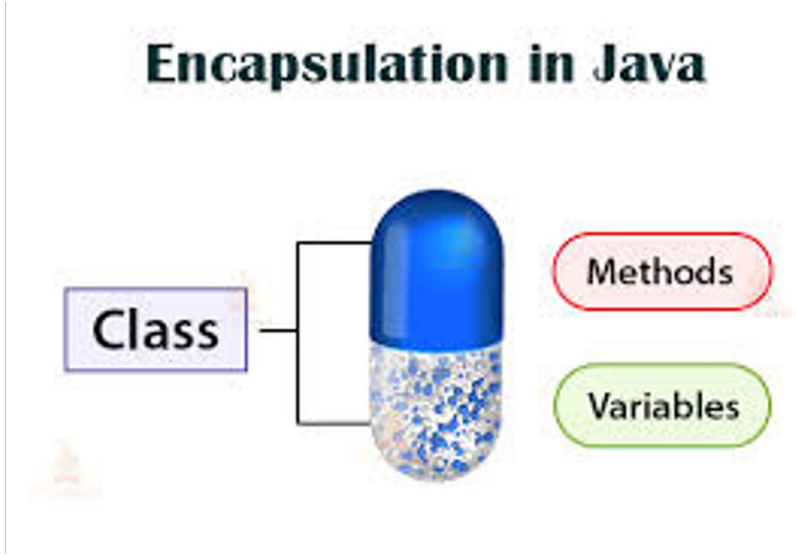
ENCAPSULATION

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

WHAT IS ENCAPSULATION?

- Encapsulation is the ability of an object to hide its data and methods.
- A fundamental principle of OOP (Object Oriented Programming).

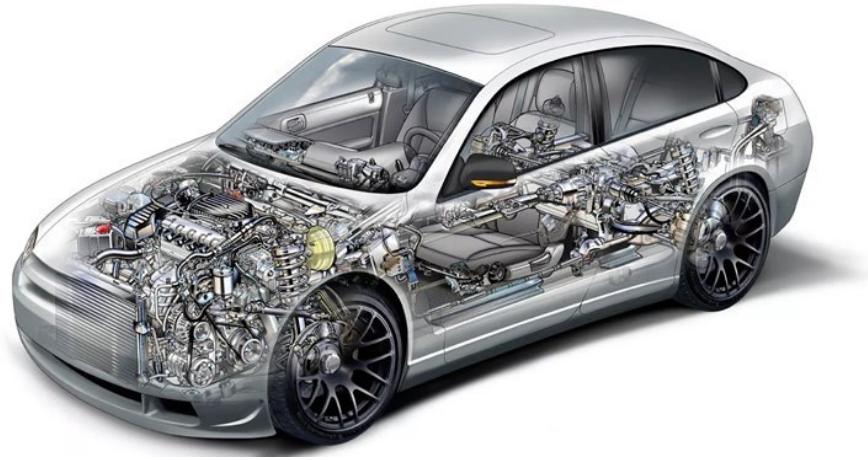


DERIVED CLASS CAN RESTRICT VISIBILITY?

- **Private** – Protected and public members of base class are private in derived class.
- **Protected** – Protected and public members of base class are protected in derived class.
- **Public** – Protected and public members of base class are protected and public in derived class.
- Private members of base class are NOT visible in derived class.

ENCAPSULATION

```
public class Car {
   private int year;
   public String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   public int getYear(){
      return year;
   }
}
```

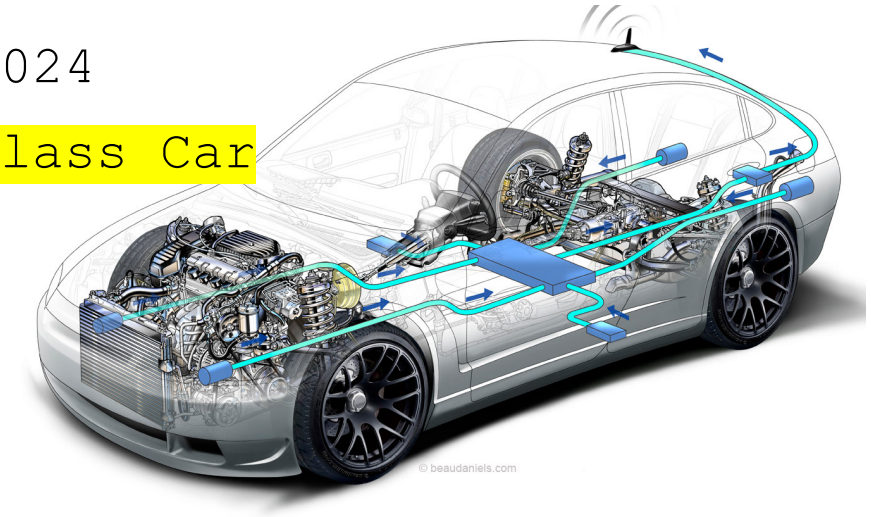


EXERCISE

- Modify the tester program. Make a object **D** of type **Car**.
- Get the value of **year** from object **D** and print it.
- Modify the value of **year** (try it!)
- Modify the value of **model**.

```
public static void main(String [] args){
   Car D = new Car();
   System.out.println(D.getYear()); // prints 2024
   D.year = 2030;   //ERROR: private member of class Car
   D.model = "Toyota"; //ok !!
}
```

```
Car(){
    year = 2024;
    model = "Lexus";
  }
```

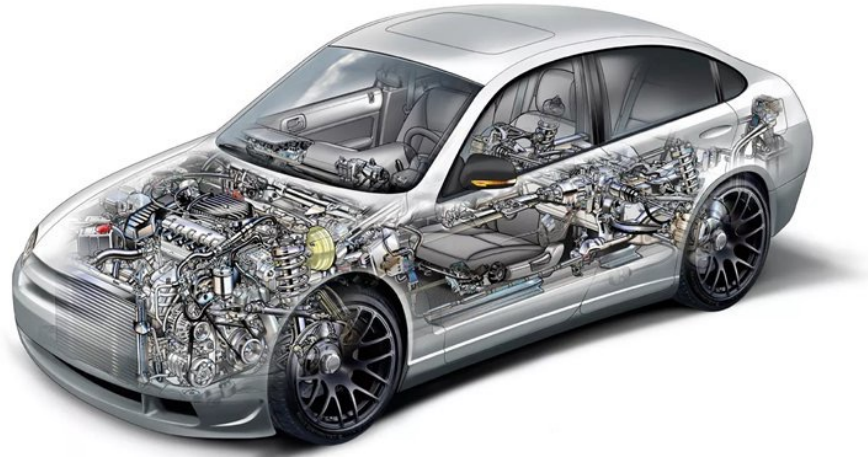


GETTERS AND SETTERS

- A **getter** is a method that extracts information.
- A **setter** is a method that inserts information.
- **Getters** and **setters** can be used even without underlying matching variables

EXAMPLE

```
public class Car {
   private int year;
   public String model;
   Car(){
      year = 2024;
      model = "Lexus";
   }
   public int getYear(){                  //getter
      return year;
   }
   public void setYear(int year){   //setter
      this.year = year;
   }
}
```

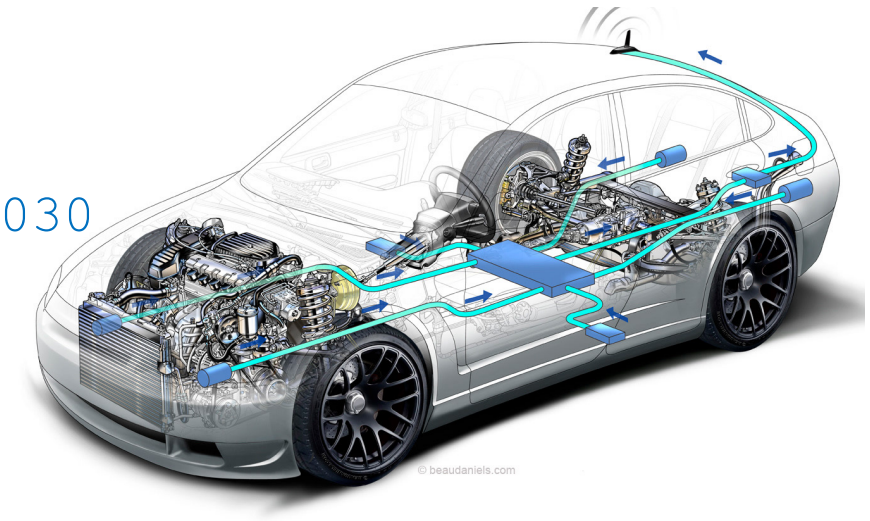


EXERCISE

- Modify the tester program. Make a object **E** of type **Car**.
- Set the value of **year** of object **E** to 2030.
- Print the value of **year** using the getter.

```
public static void main(String [] args){
   Car E = new Car();
   System.out.println(E.getYear()); // prints 2024
   E.year = 2030; //Error: private member
   E.setYear(2030);
   System.out.println(E.getYear()); // prints 2030
}
```

```
Car(){
    year = 2024;
    model = "Lexus";
  }
```



CS102 Programming II

STATIC

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

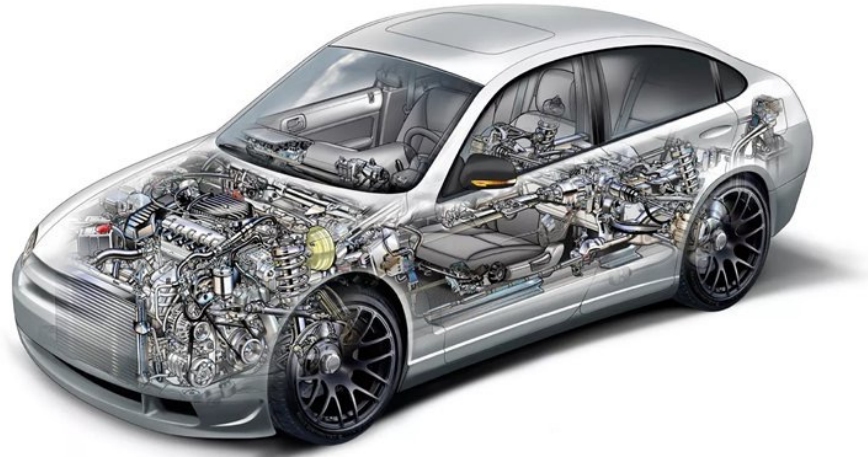
STATIC

- **static class member** is a member that belongs to the **class itself**, rather than to individual objects (instances) of that class.
- There are two main types:
  - **static attribute** (class attribute)
  - **static method** (class method)

STATIC ATTRIBUTE

```
public class Car {
  private int year;
  public String model;
  public static int numberOfCars = 0;
  Car(int year, String model){
    this.year = year;
    this.model = model;
    numberOfCars++;
  }
  public int getYear(){              //getter
    return year;
  }
  public void setYear(int year){   //setter
    this.year = year;
  }
}
```

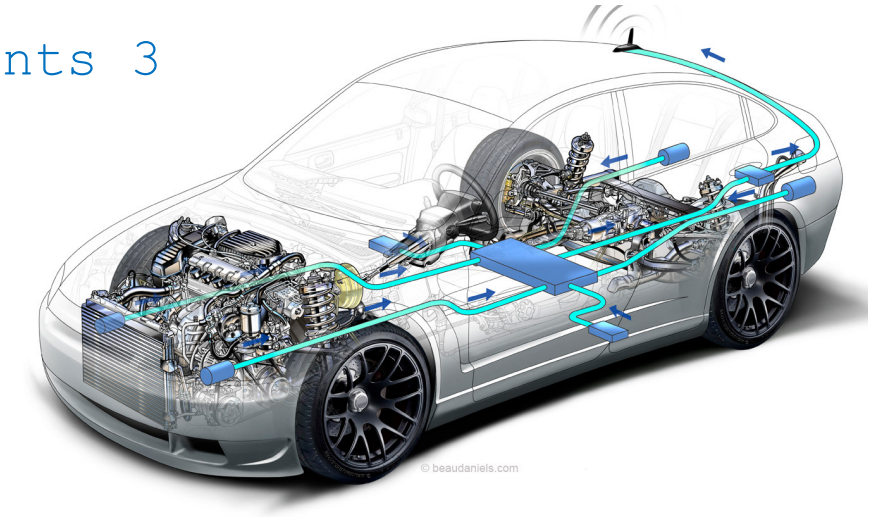
static means numberOfCars belong to class **Car** and not objects



STATIC CLASS MEMBER TESTER

```
public static void main(String [] args){
   Car C1 = new Car(2021, "Lexus");
   //numberofCars is now 1
   Car C2 = new Car(2022, "Ford 150");
   //numberofCars is now 2
   Car C3 = new Car(2026, "BMW X6");
   //numberofCars is now 3
   System.out.println(Car.numberOfCars); // prints 3
}
//We don't need an object because
//numberOfCars belongs to the class.
```

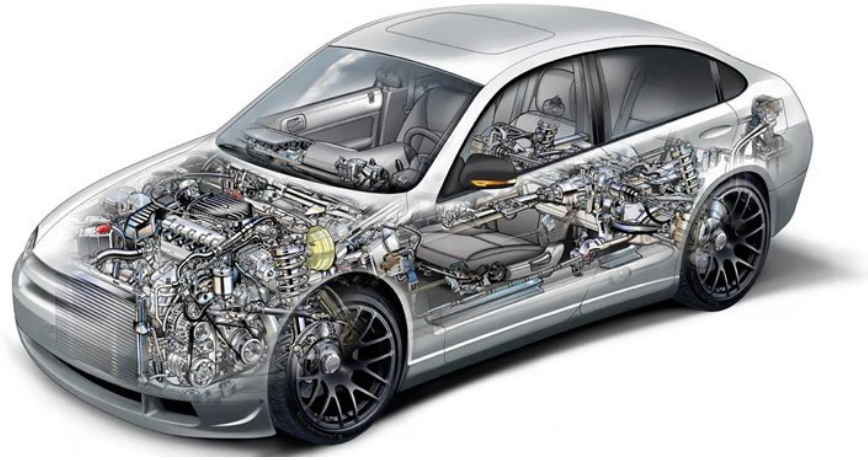
```
Car(int year, String model){
   this.year = year;
   this.model = model;
   numberOfCars++;
 }
```



STATIC METHOD

```
public class Car {
  private int year;
  public String model;
  private static int numberOfCars = 0;
  Car(int year, String model){
    this.year = year;
    this.model = model;
    numberOfCars++;
  }
  public int getYear(){              //getter
    return year;
  }
  public void setYear(int year){   //setter
    this.year = year;
  }
  public static int getNumberOfCars(){ //static getter
    return numberOfCars;
  }
}
```

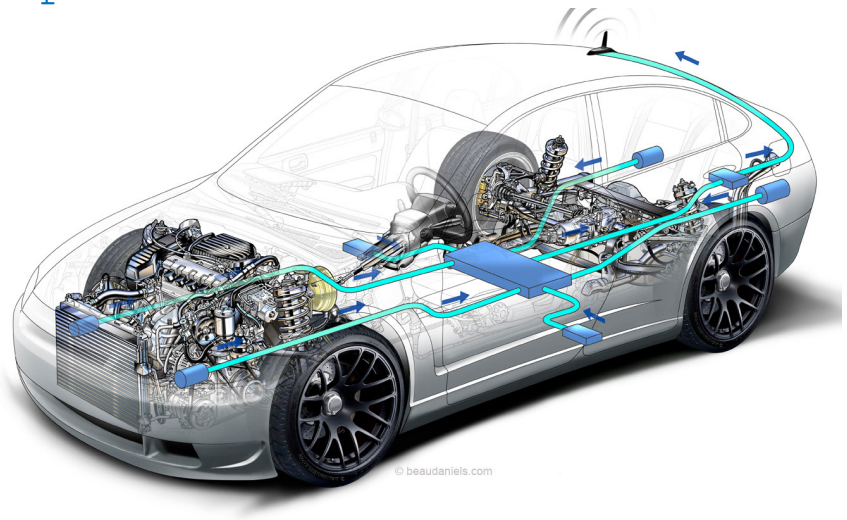
static means numberOfCars belong to class **Car** and not objects



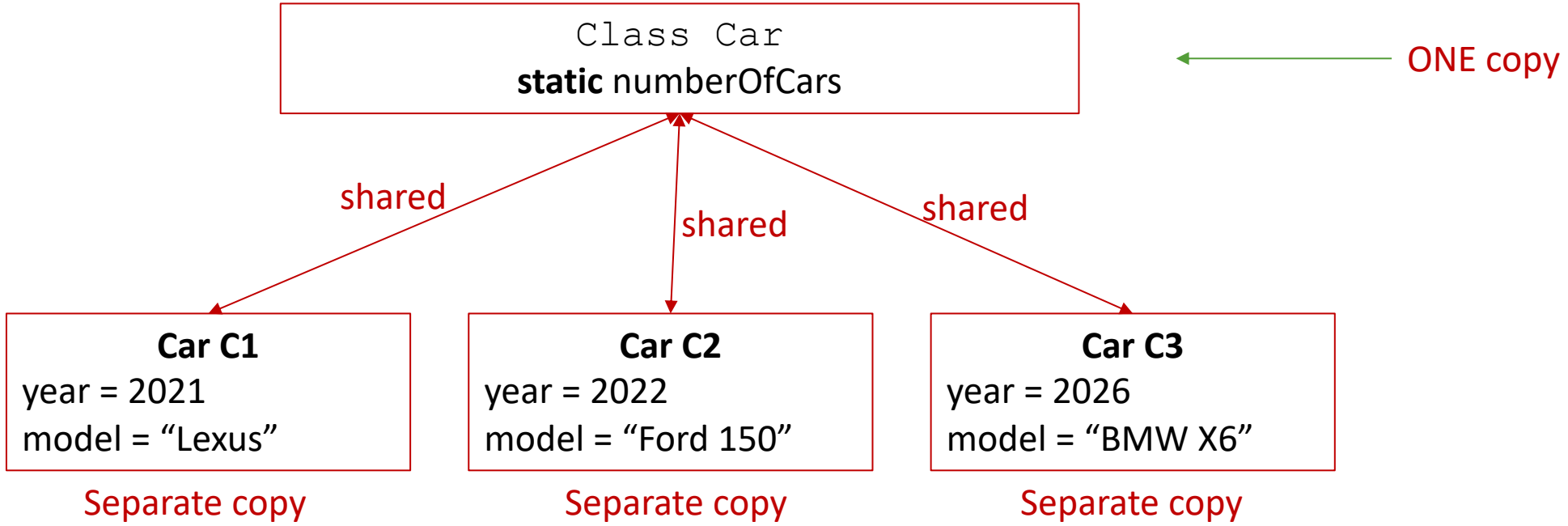
STATIC METHOD TESTER

```
public static void main(String [] args){
   Car C1 = new Car(2021, "Lexus");
   Car C2 = new Car(2022, "Ford 150");
   Car C3 = new Car(2026, "BMW X6");
   //numberofCars is now 3
   System.out.println(Car.numberOfCars); // ERROR private member
   System.out.println(Car.getNumberOfCars()); // prints 3
}
//We don't need an object because
//numberOfCars belongs to the class.
```

```
Car(int year, String model){
   this.year = year;
   this.model = model;
   numberOfCars++;
 }
```



STATIC SUMMARY



Instance member → one copy for every object.
Static member → one copy for the entire class.

CS102 Programming II

FINAL

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

FINAL

- In Java, **`final` means "cannot be changed"** after it has been assigned.

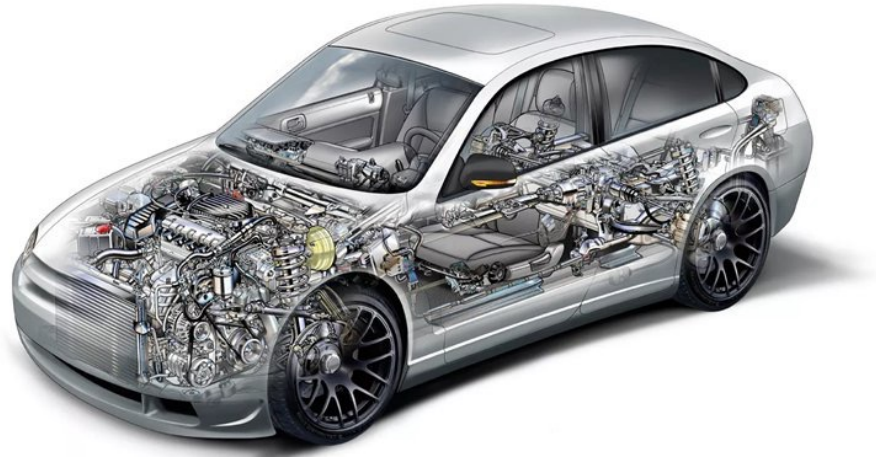
`static` → shared by the class

`final` → cannot be changed

`static final` → one shared value that cannot be changed

FINAL ATTRIBUTE

```
public class Car {
  private int year;
  public String model;
  public final int MAX_NUMBER_OF_CARS = 100;
  Car(int year, String model){
    this.year = year;
    this.model = model;
  }
  public int getYear(){              //getter
    return year;
  }
  public void setYear(int year){   //setter
    this.year = year;
  }
}
```

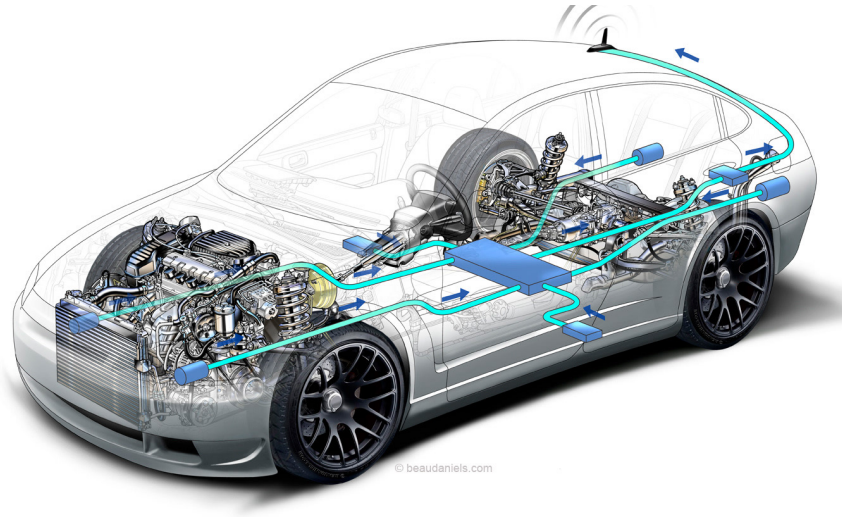


FINAL ATTRIBUTE CHANGE TESTER

```
public static void main(String [] args){
   Car C1 = new Car(2021, "Lexus");
   Car C2 = new Car(2022, "Ford 150");
   Car C3 = new Car(2026, "BMW X6");
   System.out.println(C3.MAX_NUMBER_OF_CARS ); // prints 100
   C3.MAX_NUMBER_OF_CARS = 10; // Changing final is NOT ALLOWED
}
```

//It is commonly used for **constants**. Example:

```
//public static final int MAX_SPEED = 120;
```



CS102 Programming II

TOSTRING()

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

THE TOSTRING METHOD

- The special method toString:

  Tells Java how to convert your object into a String as needed.

  Is called when an object is printed or concatenated to a String.

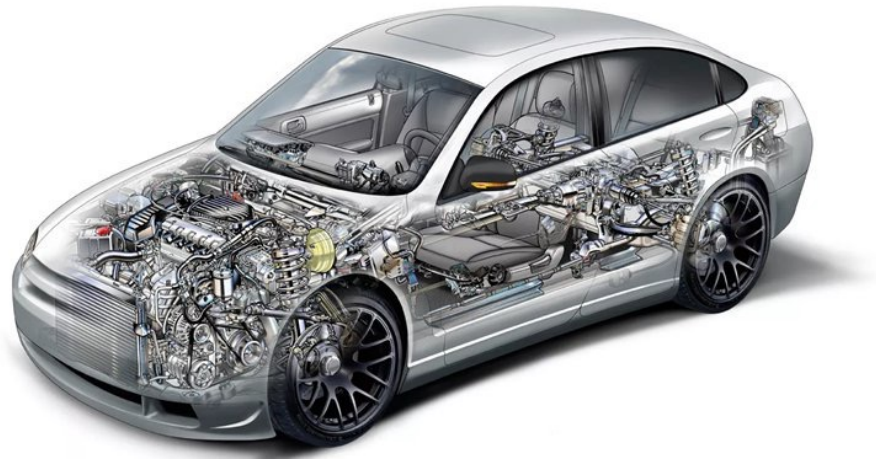
- Every class has a toString, even if it isn't in your code.

  The default toString returns the class's name followed by a hexadecimal (base-16) number:

  ```
  "Car@9e8c34"
  ```

TOSTRING

```
public class Car {
   int year;
   String model;
   Car(int year, String model){
      this.year = year;
      this.model = model;
   }
   public String toString(){
      return "(" + year + ", " + model + ")" ;
   }
}
```



TOSTRING TESTER

- Demostrate usage of toString()

```
public static void main(String [] args){
   Car C1 = new Car(2021, "Lexus");
   Car C2 = new Car(2022, "Ford 150");
   System.out.println(C1.toString());  // prints: (2021, Lexus)

   System.out.println(C2);  // prints: (2022, Ford 150)
}
```

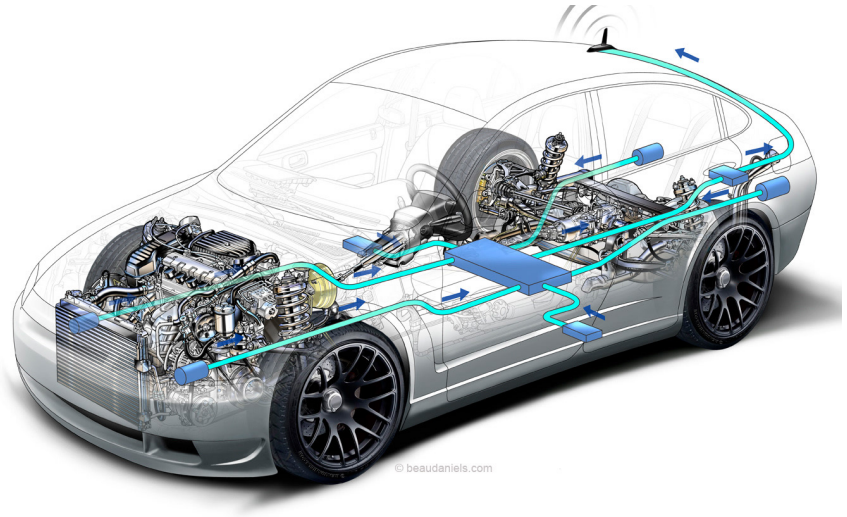
//toString() returns a string. So we can call the toString through the object

```
//C1.toString();
```

//printing an object also calls the toString() method (Override) of that object.

```
//C1;
```

//We study (Override) later.



CS102 Programming II

COMPOSITION

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

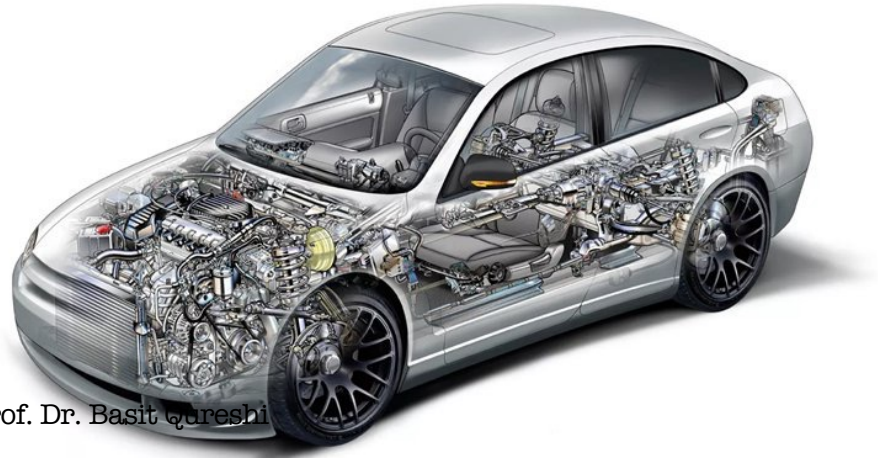
COMPOSITION

- A class can have references to objects of other classes as members
- Sometimes referred to as a *has-a* relationship

51

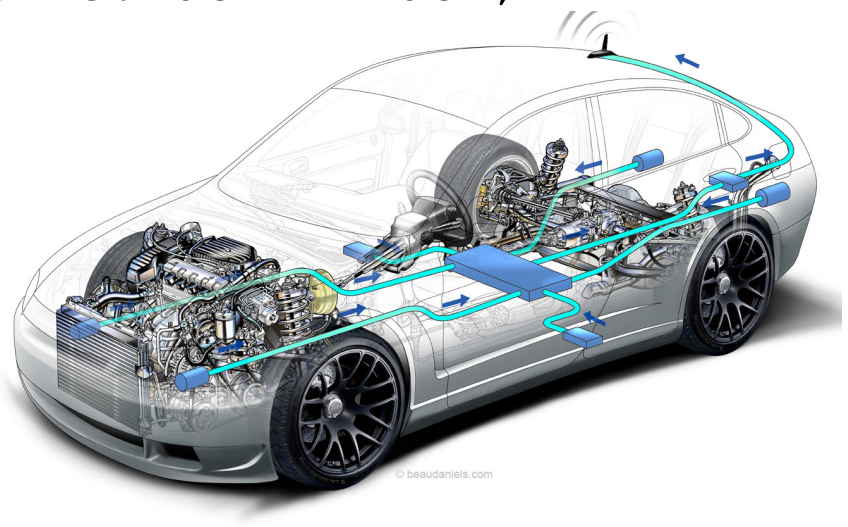
TOSTRING

```
public class Car {
   int year;
   String model;
   Engine e;
   Car(){ //initialize attributes
   }
   Car(int year, String model){
      this.year = year;
      this.model = model;
   }
}
```



```
public class Engine {
   int hp;
   String fuel;
   Engine() {//initialize attributes


   }
   Engine(int hp, String fuel){
      this.hp = 10;
      this.fuel = fuel;
   }
}
```



TOSTRING

```
public class Car {
   int year;
   String model;
   Engine e;
   Car(int year, String model){
      this.year = year;
      this.model = model;
   }
}
```

```
public class Engine {
   int hp;
   String fuel;
   Engine(int hp, String fuel){
      this.hp = hp;
      this.fuel = fuel;
   }
}
```

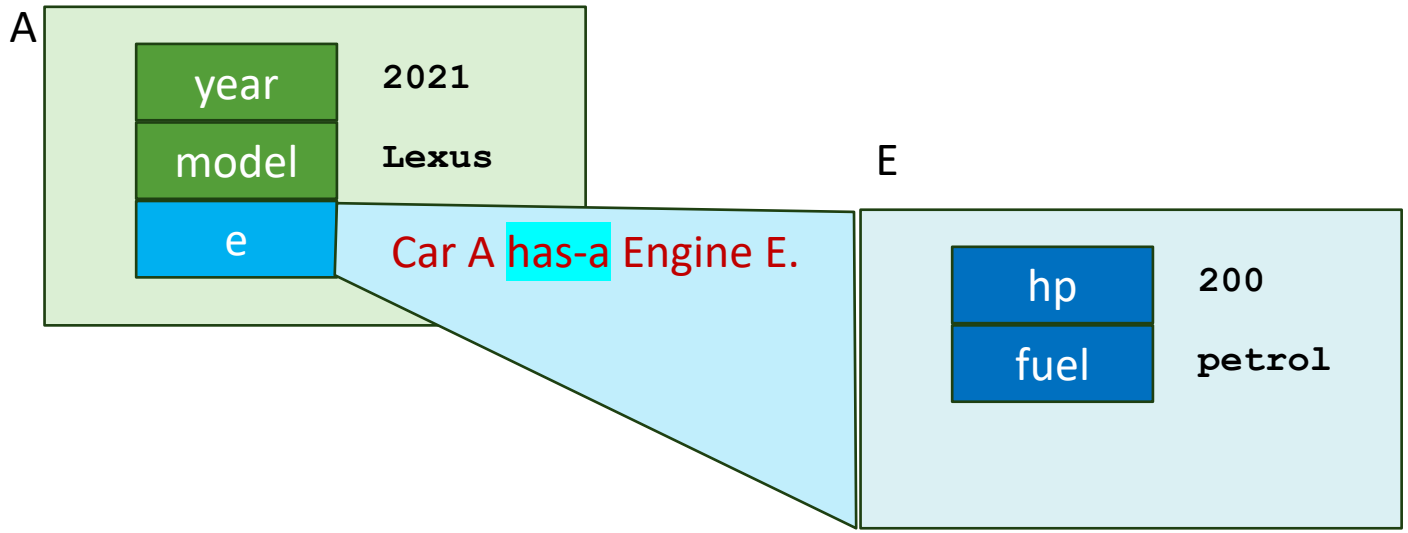
Every **Car** has-a **Engine**.

Engine is a class. **e** is object of class **Engine**, which is a attribute of class **Car**.

COMPOSITION TESTER

- Make a Car object and initialize the composed object of Engine class

```
public static void main(String [] args){
   Car A = new Car(2021, "Lexus");
   Engine E = new Engine(200, "petrol");
  A.e = E;
  }
```



Methods of class Engine can be available to object A.e

CS102 Programming II

EQUALS AND INSTANCEOF

Prof. Dr. Basit Qureshi
PhD FHEA SMIEEE MACM
www.drbasit.org

جامعة الأمير سلطان
PRINCE SULTAN UNIVERSITY

CLASS **Object**

- In java, all types of objects have a **superclass** named **`Object`**.
  - Every class implicitly `extends Object.`

    // we study extends later!!
- The Object class defines several methods:
  - `public String toString()`

    Used to print the object.
  - `public boolean equals(Object other)`

    Compare the object to any other for equality.

CLASS Object

- You can store any object in a variable of type `Object`.

```
Object o1 = new Car(2020, "G.M.C.);
Object o2 = "hello there";
Object o3 = new Scanner(System.in);
```

- An `Object` variable only knows how to do general things.

```
String s = o1.toString();          // ok
int len = o2.length();             // error
String line = o3.nextLine();       // error
```

- You can write methods that accept an `Object` parameter.

```
public void example(Object o) {
    if (o != null) {
        System.out.println("You passed " + o.toString());
    }
}
```

CLASS Object

- The == operator does not work well with objects.
  - == compares references to objects, not their state.
  - Example:

```
Car c1 = new Car(2020, "GMC");
Car c2 = new Car(1999, "Ford");
if (c1 == c2) {    // false
    System.out.println("equal");
}
```

equals

- The `equals` method compares the state of objects.

```
if (str1.equals(str2)) {
    System.out.println("the strings are equal");
}
```

- But if you write a class, its `equals` method behaves like `==`

```
if (c1.equals(c2)) {    // false :-(
    System.out.println("equal");
}
```

This is the behavior we inherit from class **`Object`**. Java doesn't understand how to compare `Car` by default.

TYPE-CASTING OBJECTS

- Solution: *Type-cast* the object parameter to a `Car`.

```
public boolean equals(Object o) {
    Car other = (Car) o; ←type casting
    return year == o.year && model == o.model;
}
```

- Casting objects is different than casting primitives.
  - Tells the compiler to *assume* that `o` refers to a `Car` object.

instanceof

```
if (variable instanceof type) {
    statement(s);
}
```

- Asks whether a variable refers to an object of a given type.
  - Used as a `boolean` test.
  - Examples:

```
String s = "hello";
Car c1 = new Car();
```

expression	result
`s instanceof Car`	`false`
`s instanceof String`	`true`
`c1 instanceof Car`	`true`
`c1 instanceof String`	`false`
`c1 instanceof Object`	`true`
`s instanceof Object`	`true`
`null instanceof String`	`false`

instanceof

```
public boolean equals(Object o) {
    if (o instanceof Car) {
        // o is a Car; cast and compare it
        Car other = (Car) o;
        return year==o.year && model == o.model;
    } else {
        // o is not a Car; cannot be equal
        return false;
    }
}
```