JAVA PROGRAMMING BASICS

Module 2: Java Object-oriented Programming

Training program

1. Classes and Instances
2. The Methods
3. The Constructors
4. Static elements
5. Initialization sections
6. Package
7. Inheritance and Polymorphism
8. Abstract classes and interfaces
9. String processing
10. Exceptions and Assertions
11. Nested classes
12. Enums
13. Wrapper classes for primitive types
14. Generics
15. Collections
16. Method overload resolution
17. Multithreads
18. Core Java Classes
19. Object Oriented Design

Module contents

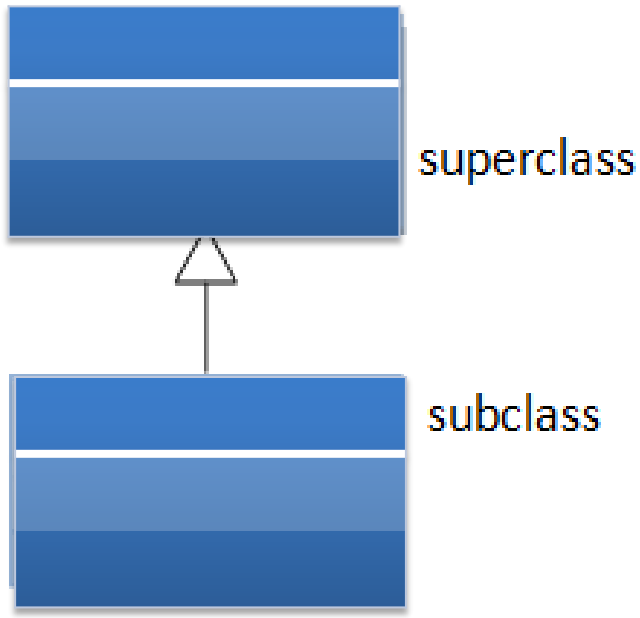
- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier
  - The protected access modifier

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

The Inheritance 1/5

- Inheritance is the concept of a child class (sub class) automatically inheriting the variables and methods defined in its parent class (super class).
- Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.



The Inheritance 2/5

- Benefits of Inheritance in OOP : Reusability
- – Once a behavior (method) is defined in a super class, that behavior is automatically inherited by all subclasses
- – Once a set of properties (fields) are defined in a super class, the same set of properties are inherited by all subclasses
- – A subclass only needs to implement the differences between itself and the parent.

The Inheritance 3/5

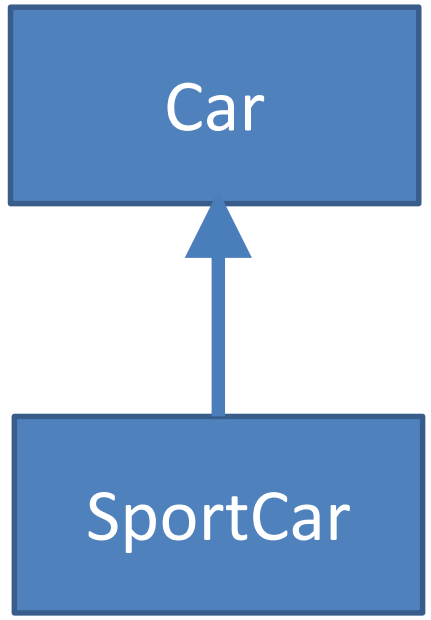
- To derive a child class, we use the **extends** keyword.

```
1. public class Car {
2. //...
3. }

4.  class SportCar extends Car{
5. //...
6. }
```

Parent class

Child class



The Inheritance 4/5

- A subclass inherits all of the “public” and “protected” members (fields or methods) of its parent, no matter what package the subclass is in
- If the subclass is in the same package as its parent, it also inherits the package-private members (fields or methods) of the parent

The Inheritance 5/5

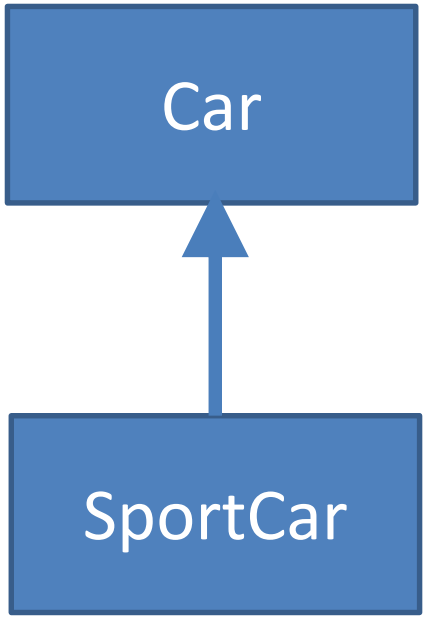
```
1.  class Car {
2.      private int maxSpeed=180;
3.      public int getMaxSpeed() {
4.          return maxSpeed;
5.      }
6.  }
7.  class SportCar extends Car{
8.  }
9.  public class Main {
10.     public static void main(String[] arg) {
11.         SportCar myCar = new SportCar();
12.         myCar.getMaxSpeed();
13.     }
14. }
```

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier
  - The protected access modifier

Inheritance and "is-a" relationship 1/2

- IS-A is a way of saying : This object is a type of that object.



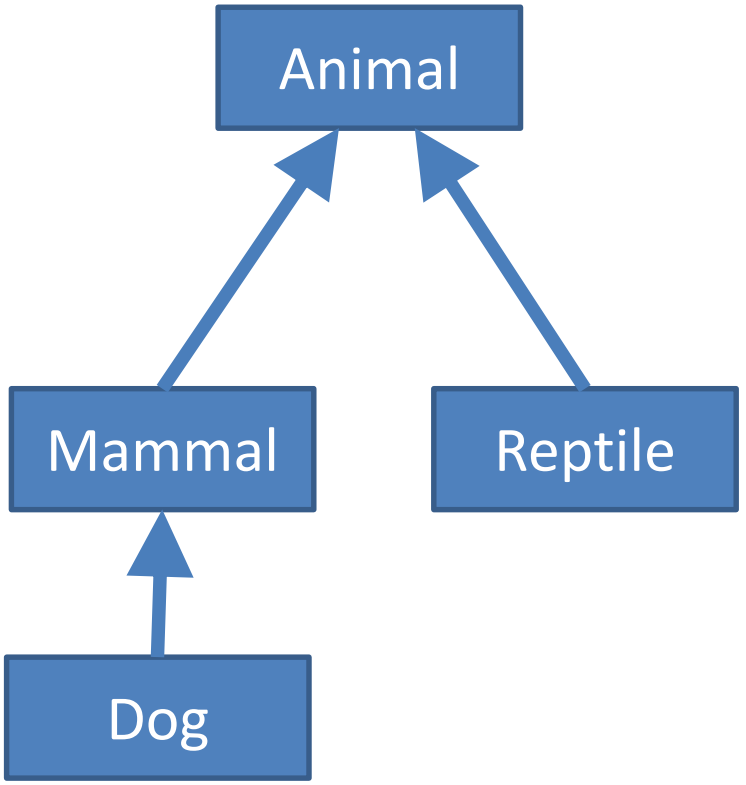
Inheritance and "is-a" relationship 2/2

```
1.  public class Animal{
2.  //...
3.  }

4.  class Mammal extends Animal{
5.  //...
6.  }

7.  class Reptile extends Animal{
8.  //...
9.  }

10. class Dog extends Mammal{
11. //...
12. }
```



Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Method overriding

- An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.

Method overriding

```
1.  class Car {
2.      public void testMethod(){
3.          System.out.println("Car");
4.      }
5.  }
6.  class SportCar extends Car{
7.      @Override
8.      public void testMethod(){
9.          System.out.println("SportCar");
10.     }
11. }
12. public class Main {
13.     public static void main(String[] arg) {
14.         Car myCar1 = new Car();
15.         SportCar myCar2 = new SportCar();
16.         myCar1.testMethod();
17.         myCar2.testMethod();
18.     }
19. }
```

Console output:

Car

SportCar

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Fields hiding

- Within a class, a field that has the same name as a field in the superclass hides the superclass's field, even if their types are different.

- Don't recommend hiding fields as it makes code difficult to read

Fields hiding

```
1.  class Car {
2.      protected int maxSpeed = 160;
3.      public void testMethod(){
4.          System.out.println(maxSpeed);
5.      }
6.  }
7.  class SportCar extends Car{
8.      protected int maxSpeed = 280;
9.      public void testMethod2(){
10.         System.out.println(maxSpeed);
11.     }
12. }
13. public class Main {
14.     public static void main(String[] arg) {
15.         SportCar myCar2 = new SportCar();
16.         myCar2.testMethod();
17.         myCar2.testMethod2();
18.     }
19. }
```

Console output:

160
160
280

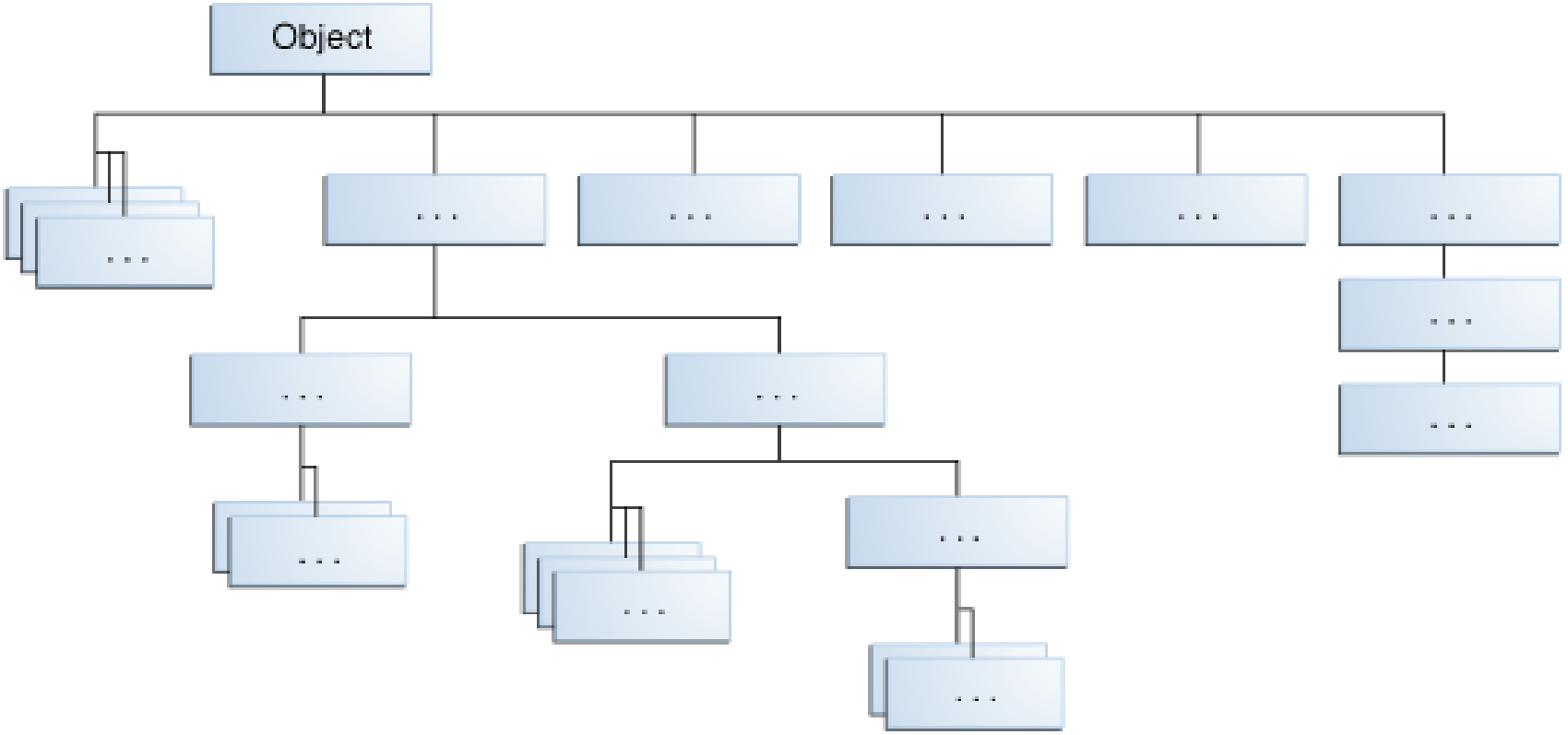
Hiding fields is discouraged!!!

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Class Object 1/3

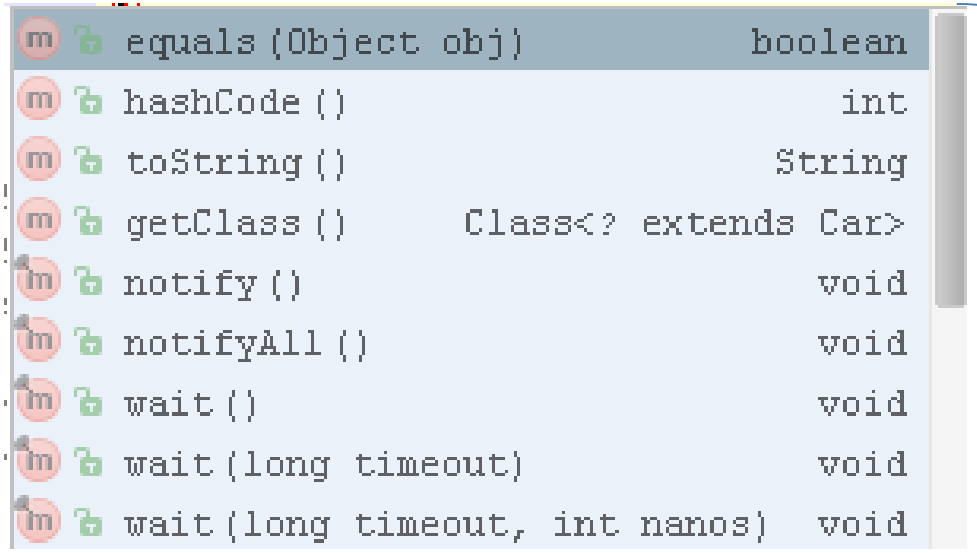
- At the top of the hierarchy, Object is the most general of all classes



Class Object 2/3

```
class MyClass {
}
public class Main {
    public static void main(String[] arg) {
        MyClass my = new MyClass();
        my.
    }
}
```

Class without any declared methods or fields

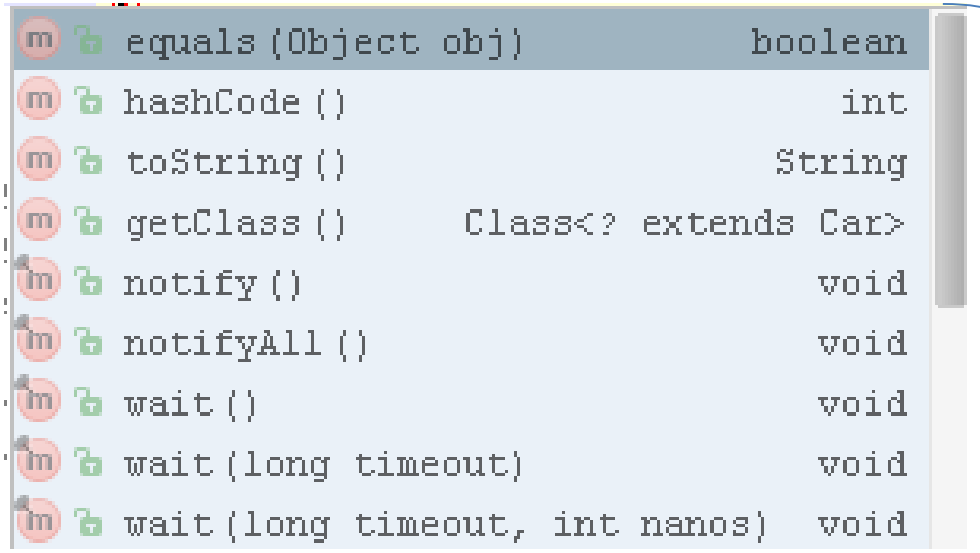


The inherited Methods from Class Object

Class Object 3/3

```
class MyClass extends Object {
}
public class Main {
    public static void main(String[] arg) {
        MyClass my = new MyClass();
        my.
    }
}
```

Explicit **extends** Object



The inherited Methods from Class Object

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Object methods: toString(), equals(), hashCode() 1/12

Modifier and Type	Method and Description
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
String	toString() Returns a string representation of the object.

Object methods: toString(), equals(), hashCode() 2/12

Modifier and Type	Method and Description
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
void	wait() Causes the current thread to wait until another thread invokes the **notify()** method or the **notifyAll()**method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the **notify()** method or the**notifyAll()** method for this object, or a specified amount of time has elapsed.

Object methods: toString(), equals(), hashCode()... 3/12

- public String toString()
- Returns a string representation of the object.

```
1. class Car {
2. }
3. public class Main {
4.     public static void main(String[] arg) {
5.         Car myCar = new Car();
6.         System.out.println(myCar);
7.     }
8. }
```

Result: com.brainacad.oop1.testclass.Car@1e893df

Object methods: toString(), equals(), hashCode()... 4/12

```
1.  class Car {
2.      @Override
3.      public String toString(){
4.          return "This is Car";
5.      }
6.  }
7.  public class Main {
8.      public static void main(String[] arg) {
9.          Car myCar = new Car();
10.         System.out.println(myCar);
11.     }
12. }
```

Result: This is Car

Object methods: toString(), equals(), hashCode()... 5/12

- public boolean equals(Object obj)
- Indicates whether some other object is "equal to" this one.
- The equals method for class Object implements returns true if x and y refer to the same object

  (x == y has the value true).

Object methods: toString(), equals(), hashCode()... 6/12

```
1.  class Car {
2.  private long id;
3.      public Car(long id){
4.          this.id = id;
5.      }
6.  }
7.  public class Main {
8.      public static void main(String[] arg) {
9.          Car myCar1 = new Car(12345);
10.         Car myCar2 = new Car(12345);
11.         Car myCar3 = myCar1;
12.         System.out.println(myCar1.equals(myCar2));
13.         System.out.println(myCar1.equals(myCar3));
14.     }
15. }
```

Console output:

false

true

Object methods: toString(), equals(), hashCode() 7/12

- The equals method implements an equivalence relation on non-null object references:

- **It is *reflexive***
- **It is *symmetric***
- **It is *transitive***
- **It is *consistent***

Object methods: toString(), equals(), hashCode()... 8/12

```
1.  class Car {
2.  private long id;
3.      public Car(long id){
4.          this.id = id;
5.      }
6.      @Override
7.      public boolean equals(Object o) {
8.          if (this == o) return true;
9.          if (o == null || getClass() != o.getClass()) return false;
10.         Car car = (Car) o;
11.         if (id != car.id) return false;
12.         return true;
13.     }
14. }
```

Console output:

true

true

Object methods: toString(), equals(), hashCode() 9/12

- public int hashCode()
- Returns a hash code value for the object

- A **hash function** is any function that can be used to map digital data of arbitrary size to digital data of fixed size. The values returned by a hash function are called **hash values**, **hash codes**, **hash sums**, or simply **hashes**.

Object methods: toString(), equals(), hashCode() 10/12

- public int hashCode()
- Returns a hash code value for the object.

```
1.  public class Main {
2.      public static void main(String[] arg) {
3.          Car myCar1 = new Car(12345,180);
4.          Car myCar2 = new Car(12345,180);
5.          Car myCar3 = new Car(44444,120);
6.          System.out.println(myCar1.hashCode());
7.          System.out.println(myCar2.hashCode());
8.          System.out.println(myCar3.hashCode());
9.      }
10. }
```

Console output:

28027784

25853693

26680060

Object methods: toString(), equals(), hashCode() 11/12

```
1.  class Car {
2.      private long id;
3.      private int maxSpeed;

4.      public Car(long id,int maxSpeed) {
5.          this.id = id;
6.          this.maxSpeed = maxSpeed;
7.      }

8.      @Override
9.      public int hashCode() {
10.         int result = (int) (id ^ (id >>> 32));
11.         result = 31 * result + maxSpeed;
12.         return result;
13.     }
14. }
```

Console output:

382875

382875

1377884

Object methods: toString(), equals(), hashCode()

```
1.  public class PhoneNumber {
2.      private int areaCode;
3.      private int prefix;
4.      private int lineNumber;
5.      //...
6.      @Override public int hashCode() {
7.          int result = 17;
8.          result = 31 * result + areaCode;
9.          result = 31 * result + prefix;
10.         result = 31 * result + lineNumber;
11.         return result;
12.     }
13. }
```

Object methods: toString(), equals(), hashCode() 12/12

- If two objects are equal according to the *equals(Object)* method, then calling the *hashCode* method on each of the two objects must produce the same integer result.

- **If you override the *equals()*, you MUST also override *hashCode()*.**

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

The "super" keyword

- Usage of java super Keyword

- super() is used to invoke immediate parent class constructor
- super is used to invoke immediate parent class method
- super is used to refer immediate parent class instance variable

The "super" keyword

```
1.  class Car {
2.      public String toString(){
3.          return "This is Car";
4.      }
5.  }
6.  class SportCar extends Car{
7.      public String toString(){
8.          return super.toString() + ":SportCar";
9.      }
10. }
11. public class Main {
12.     public static void main(String[] arg) {
13.         Car myCar1 = new Car();
14.         SportCar myCar2 = new SportCar();
15.         System.out.println(myCar1);
16.         System.out.println(myCar2);
17.     }
18. }
```

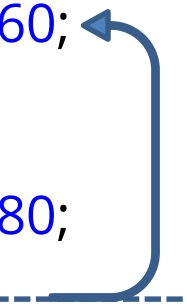
Console output:

This is Car

This is Car:SportCar

The "super" keyword

```
1.  class Car {
2.      protected int maxSpeed = 160;
3.  }
4.  class SportCar extends Car{
5.      protected int maxSpeed = 280;
6.      public void testSuper(){
7.          System.out.println(super.maxSpeed);
8.          System.out.println(this.maxSpeed);
9.      }
10. }
11. public class Main {
12.     public static void main(String[] arg) {
13.         SportCar myCar2 = new SportCar();
14.         myCar2.testSuper();
15.     }
16. }
```



Console output:

160

280

Hiding fields is discouraged!!!

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Covariant return types

```
1.  class Car {
2.      public Car getNewCar(){
3.          return new Car();
4.      }
5.  }
6.   class SportCar extends Car{
7.      @Override
8.      public SportCar getNewCar(){
9.          return new SportCar();
10.     }
11. }
```

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Constructors chaining 1/4

- Calling one constructor from other is called **Constructor chaining in Java**
- Constructors can call each other automatically or explicitly using this() and super() keywords.
- this() denotes a no argument constructor of same class and super() denotes a no argument or default constructor of parent class

Constructors chaining 2/4

```
1.  class Car {
2.      public Car(){
3.          System.out.println("Car Constructor");
4.      }
5.  }
6.  class SportCar extends Car{
7.      public SportCar(){
8.          System.out.println("SportCar Constructor");
9.      }
10. }
11. public class Main {
12.     public static void main(String[] arg) {
13.         SportCar myCar = new SportCar();
14.     }
15. }
```

Console output:

Car Constructor

SportCar Constructor

Constructors chaining 3/4

```
1.  class Car {
2.      public Car(long id){
3.          System.out.println("Car Constructor");
4.      }
5.  }
6.  class SportCar extends Car{
7.      public SportCar(long id){
8.          super(id);
9.          System.out.println("SportCar Constructor");
10.     }
11. }
12. public class Main {
13.     public static void main(String[] arg) {
14.         SportCar myCar = new SportCar(1);
15.     }
16. }
```

Console output:

Car Constructor

SportCar Constructor

Constructors chaining 4/4

```
class Car {
    public Car() {
        System.out.println("Car Constructor 1");
    }
    public Car(long id){
        System.out.println("Car Constructor 2");
    }
}
class SportCar extends Car{
    public SportCar(long id){
        System.out.println("SportCar Constructor");
    }
}
public class Main {
    public static void main(String[] arg) {
        SportCar myCar = new SportCar(1);
    }
}
```

Console output:

Car Constructor 1

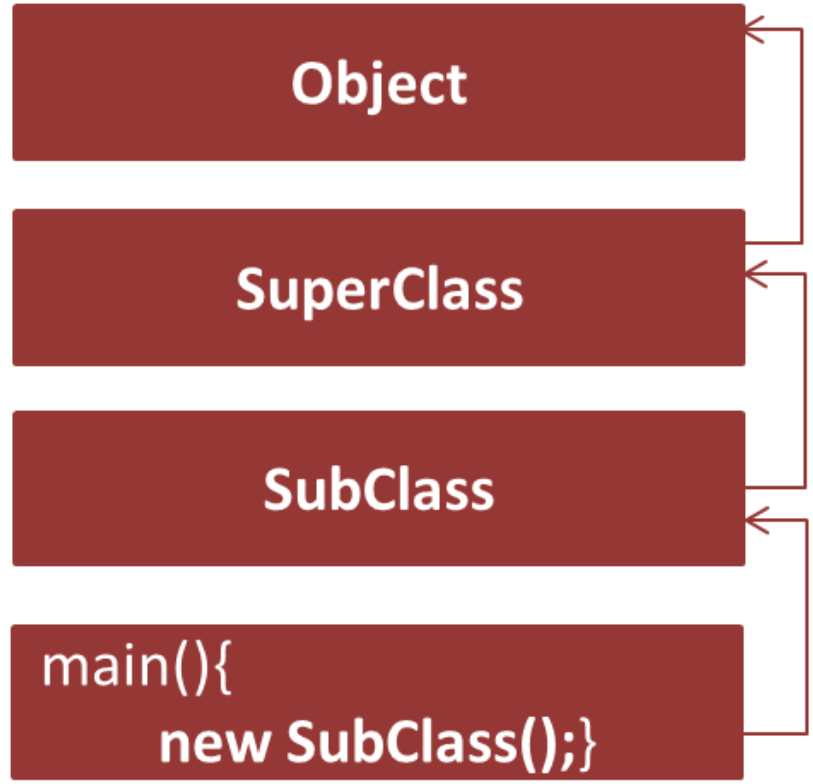
SportCar Constructor

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Type cast and conversion
  - The instanceof keyword
  - Polymorphism. Early binding & late Binding
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Initialization order and inheritance

```
1.  class SuperClass{
2.  }
3.  class SubClass extends SuperClass{
4.  }
5.  public class Start {
6.      public static void main(String[] arg){
7.          SubClass c = new SubClass();
8.      }
9.  }
```



Polymorphism. Early binding & late Binding

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

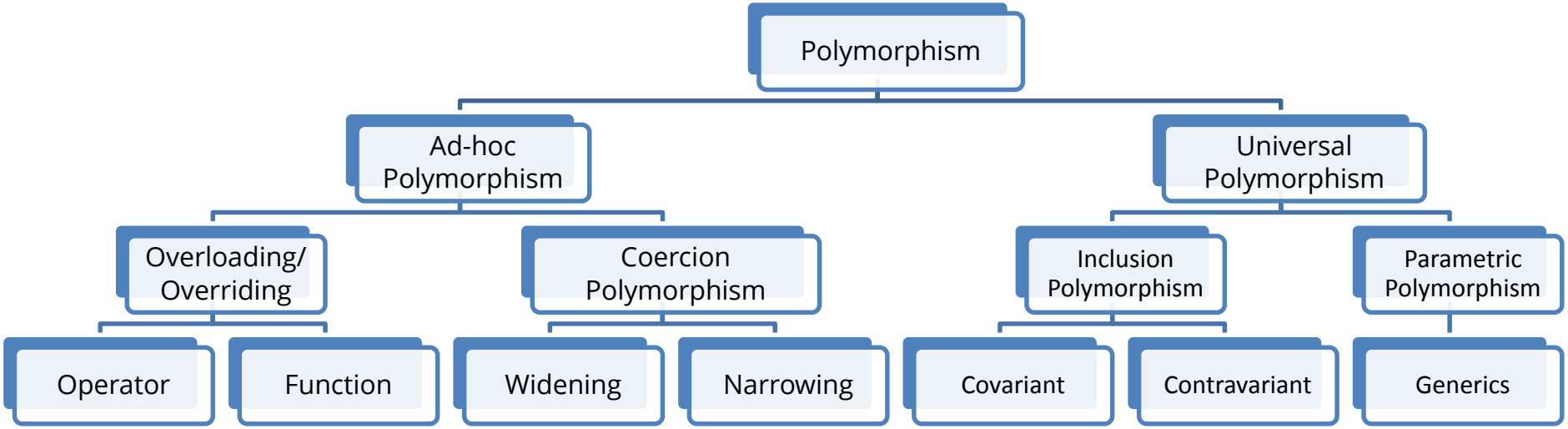
Polymorphism. Early binding & late Binding 1/5

- **Polymorphism** is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

- Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.

Polymorphism. Early binding & late Binding 2/5

Static Binding or Dynamic Binding



Polymorphism. Early binding & late Binding 3/5

```
1.  class Car {
2.  //...
3.  }
4.  class SportCar extends Car{
5.  //...
6.  }
7.  class Truck extends Car{
8.  //...
9.  }
10. public class Main {
11.     public static void main(String[] arg) {
12.         Car myCar = new Car();
13.         myCar = new SportCar();
14.         myCar = new Truck();
15.     }
16. }
```

Polymorphism. Early binding & late Binding 4/5

```
1.  class Car {
2.      public void move(){
3.          System.out.println("Car move");
4.      }
5.  }
6.  class SportCar extends Car{
7.      @Override
8.      public void move(){
9.          System.out.println("SportCar move");
10.     }
11. }
12. class Truck extends Car{
13.     @Override
14.     public void move(){
15.         System.out.println("Truck move");
16.     }
17. }
```

Polymorphism. Early binding & late Binding 5/5

- Late binding, or dynamic binding, is a computer programming mechanism in which the method being called upon an object is looked up by name at runtime.

```
1.  public class Main {
2.      public static void main(String[] arg) {
3.          Car[] myCars = {new Car(), new SportCar(), new Truck()};
4.          for(Car myCar:myCars){
5.              myCar.move();
6.          }
7.      }
8.  }
```

Console output:

Car move

SportCar move

Truck move

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Type cast and conversion 1/2

- **Java object typecasting** one object reference can be type cast into another object reference. The cast can be to its own class type or to one of its subclass or superclass types or interfaces:

- CastExpression:
  (ReferenceType)*RelationalExpression*
- *Example:*

1. Object obj = **"abcd"**;
2. String str = (String)obj;

Type cast and conversion 2/2

- It is an **ClassCastException** which occurs if you attempt to downcast a class, but in fact the class is not of that type.

```
1. public class Main {
2.     public static void main(String[] arg) {
3.         Car myCar = new SportCar();
4.         SportCar MyCar2 = (SportCar)myCar; // OK!
5.         Truck MyCar3 = (Truck)myCar; // ClassCastException!
6.     }
7. }
```

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

The instanceof keyword 1/2

- At run time, the result of the instanceof operator is true if the value of the *RelationalExpression* is not null and the reference could be cast to the *ReferenceType* without raising a ClassCastException. Otherwise the result is false.

The instanceof keyword 2/2

```
1.  public class Main {
2.      public static void main(String[] arg) {
3.          Car myCar = new SportCar();
4.          if(myCar instanceof SportCar) {
5.              System.out.println("SportCar");
6.              SportCar MyCar2 = (SportCar) myCar;
7.          }
8.          if(myCar instanceof Truck) {
9.              System.out.println("Truck");
10.             Truck MyCar3 = (Truck) myCar;
11.         }
12.     }
13. }
```

Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Objects cloning 1/7

- The **object cloning** is a way to create exact copy of an object.
- For this purpose, clone() method of Object class is used to clone an object.

Objects cloning 2/7

```
1.  public class Car {
2.      protected int maxSpeed;
3.      protected Date manufDate;
4.      public Car(Date manufDate, int maxSpeed){
5.          this.manufDate = manufDate;
6.          this.maxSpeed = maxSpeed;
7.      }
8.      @Override
9.      public String toString() {
10.         return "Max Speed = "+ maxSpeed +
11.         " Manufacture date = "+ manufDate;
12.     }
13.     public Object clone() throws CloneNotSupportedException {
14.         return super.clone();
15.     }
16. }
```

Objects cloning 3/7

```
1.  public class Main {
2.      public static void main(String[] arg) throws Exception {
3.          SimpleDateFormat sdf = new
4.          SimpleDateFormat("dd.MM.yyyy");
5.          Date dt = sdf.parse("12.09.2009");
6.          Car myCar1 = new Car(dt, 180);
7.          System.out.println(myCar1);
8.          Car myCar2 = (Car) myCar1.clone();
9.          System.out.println(myCar2);
10.     }
11. }
```

Console output:

Exception in thread "main" java.lang.CloneNotSupportedException ...

Objects cloning 4/7

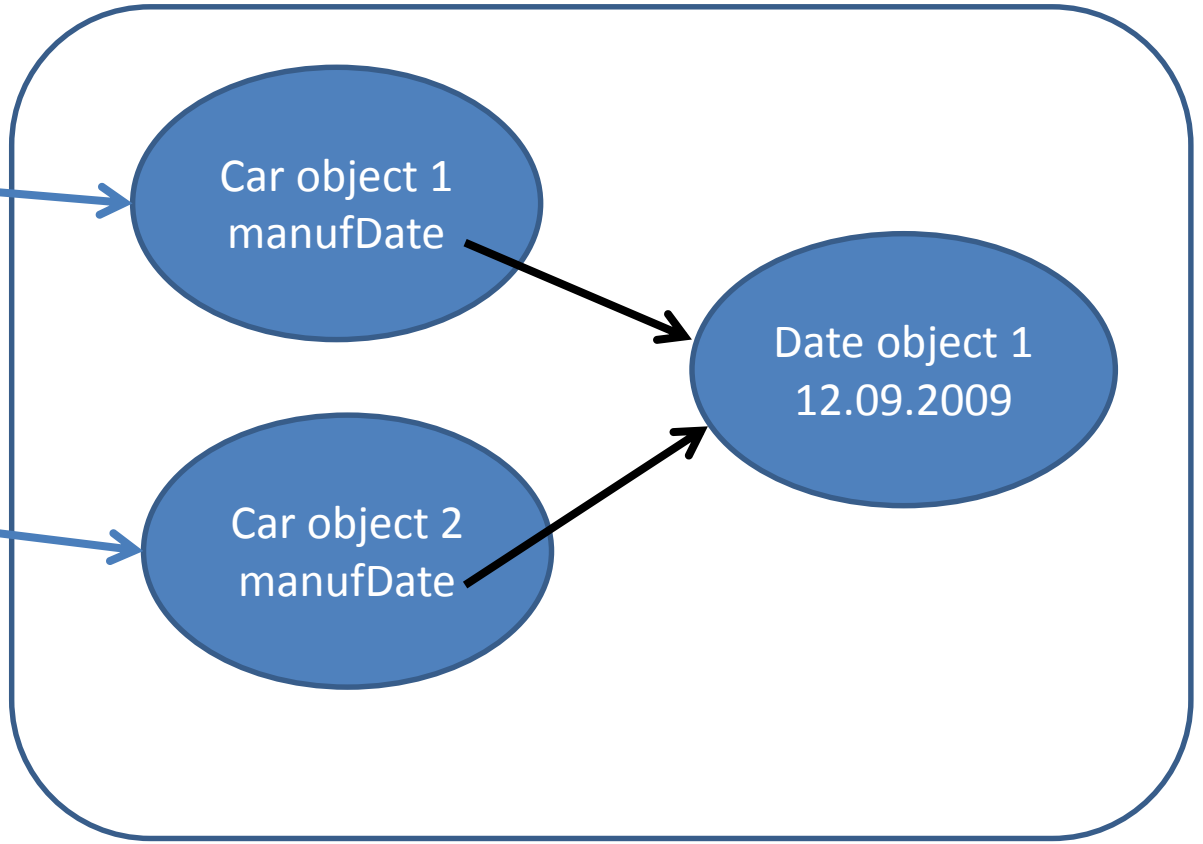
```
1.  public class Car implements Cloneable {
2.      protected int maxSpeed;
3.      protected Date manufDate;
4.      public Car(Date manufDate, int maxSpeed){
5.          this.manufDate = manufDate;
6.          this.maxSpeed = maxSpeed;
7.      }
8.      @Override
9.      public String toString() {
10.         return "Max Speed = "+ maxSpeed +
11.         " Manufacture date = "+ manufDate;
12.     }
13.     public Object clone() throws CloneNotSupportedException {
14.         return super.clone();
15.     }
16. }
```

Console output:

```
Max Speed = 180
Manufacture date =
Sat Sep 12 00:00:00
EEST 2009
Max Speed = 180
Manufacture date =
Sat Sep 12 00:00:00
EEST 2009
```

Objects cloning 5/7

- Shallow copy
- myCar1
- myCar2

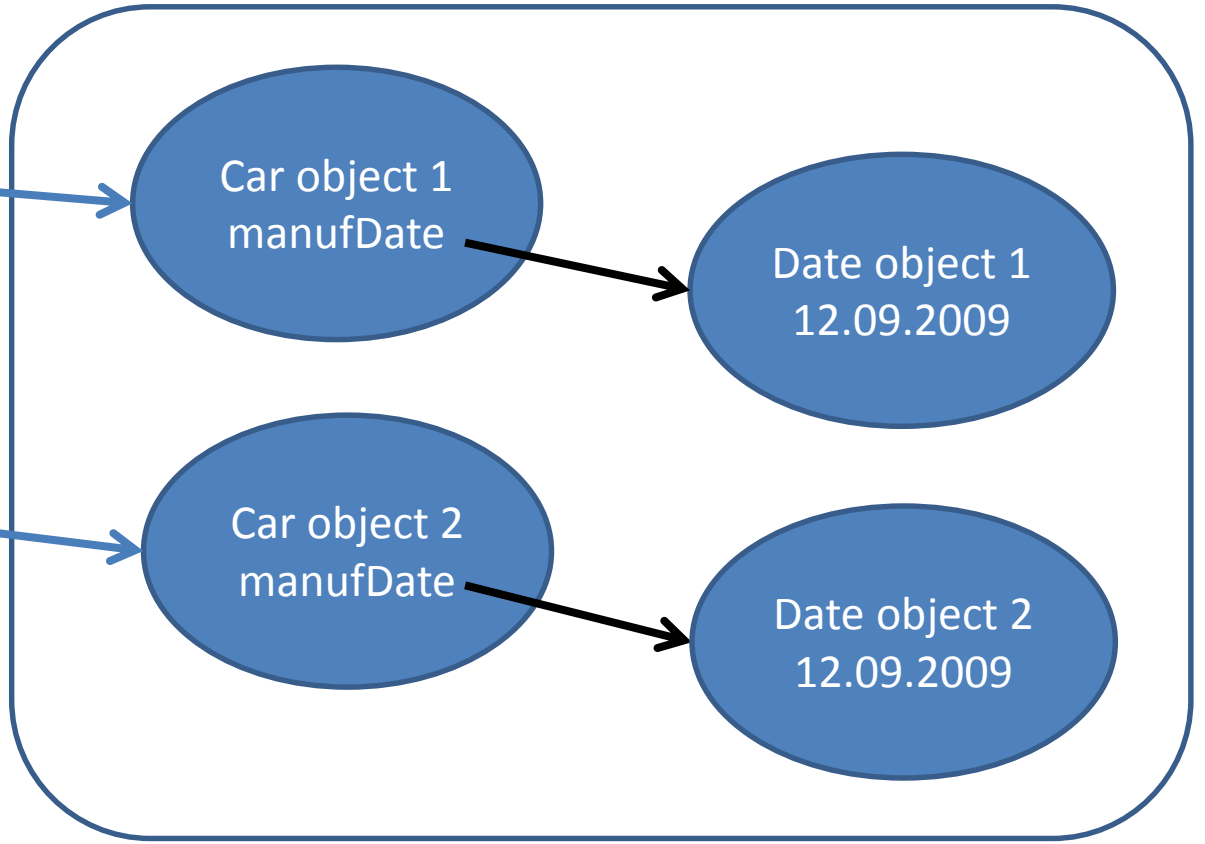


Objects cloning 6/7

```
1.  public class Car  implements Cloneable {
2.      protected int maxSpeed;
3.      protected Date manufDate;
4.      public Car(Date manufDate, int maxSpeed){
5.          this.manufDate = manufDate;
6.          this.maxSpeed = maxSpeed;
7.      }
8.      @Override
9.      public String toString() {
10.         return "Max Speed = "+ maxSpeed +
11.         " Manufacture date = "+ manufDate;
12.     }
13.     public Object clone() throws CloneNotSupportedException {
14.         Car car = (Car)super.clone();
15.         car.manufDate = (Date)this.manufDate.clone();
16.         return car;
17.     }
18. }
```

Objects cloning 7/7

- Deep copy
- myCar1
- myCar2



Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

Final class and final methods 1/3

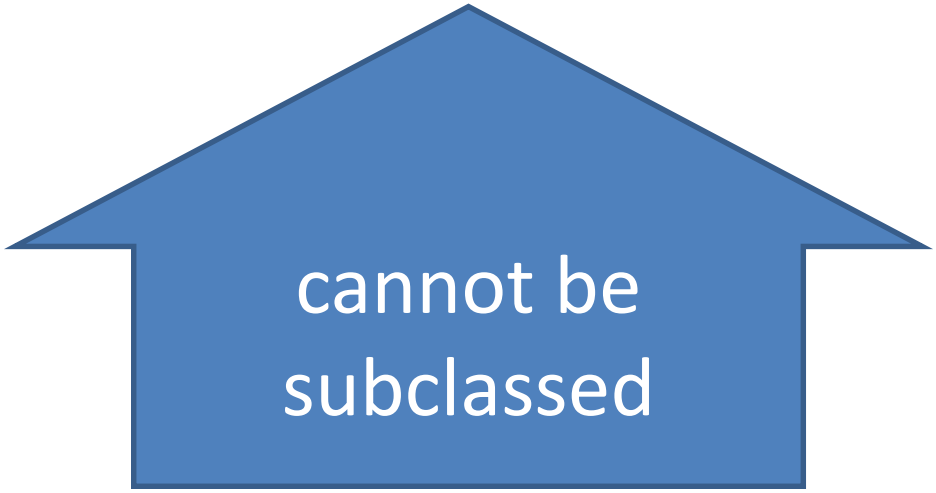
- A class that is declared **final** cannot be subclassed.
- **final class** Car { //…

  }
- You use the **final** keyword in a method declaration to indicate that the method cannot be overridden by subclasses.
- **public final void** move() {//…

  }

Final class and final methods 2/3

```
1. final class Car {
2.   //...
3. }
4. class SportCar extends Car{
5.   //...
6. }
```



Final class and final methods 3/3

- ```
  class Car {
      //...
      public final void move() {
          System.out.println("Car move");
      }
  }
  final class SportCar extends Car {
      //...
      public void move() {
          System.out.println("Car move");
      }
  }
  ```


  ```

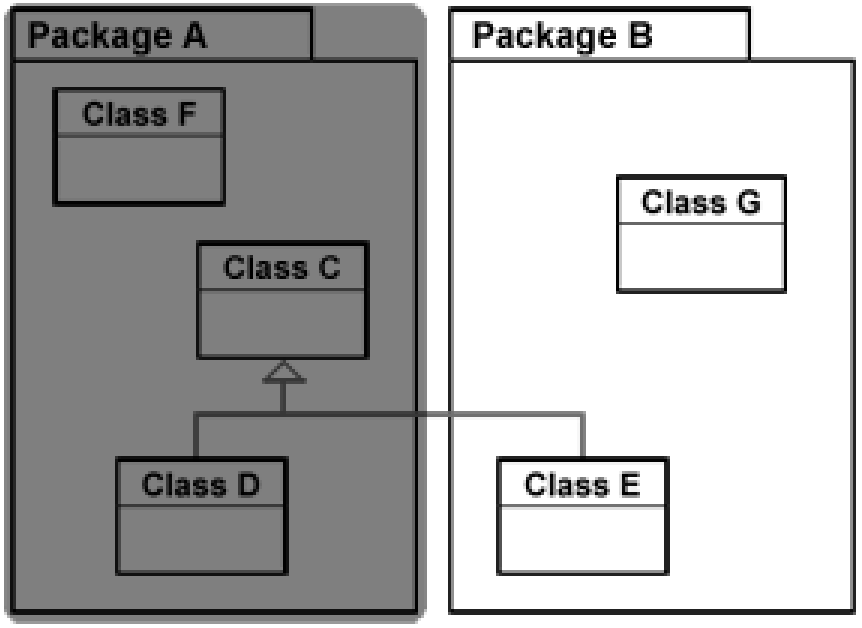
Module contents

- Inheritance and Polymorphism
  - The Inheritance
  - Inheritance and "is-a" relationship
  - Method overriding
  - Fields hiding
  - Class Object
  - Object methods: toString(), equals(), hashCode(), e.t.c.
  - The "super" keyword
  - Covariant return types
  - Constructors chaining
  - Initialization order and inheritance
  - Polymorphism. Early binding & late Binding
  - Type cast and conversion
  - The instanceof keyword
  - Objects cloning
  - Final class and final methods
  - The protected access modifier

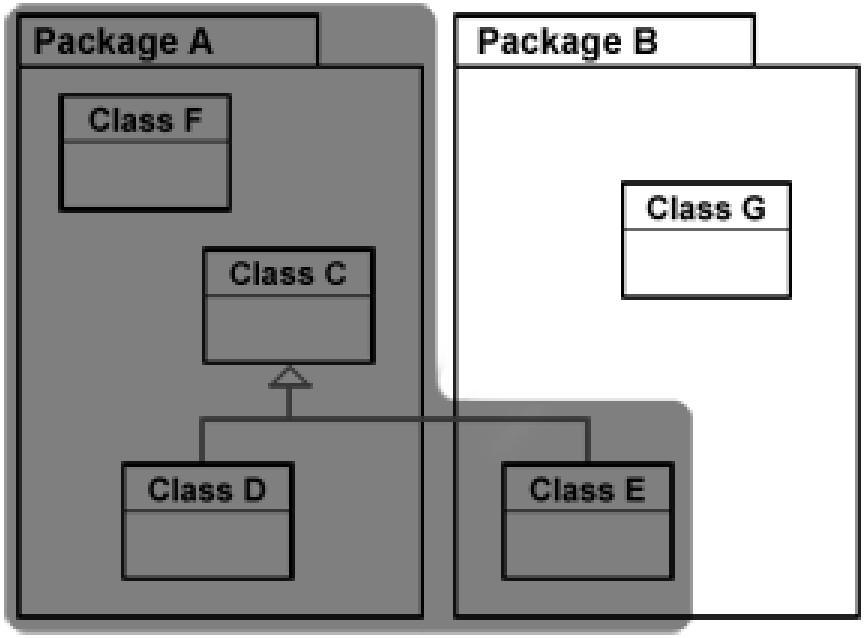
The protected access modifier 1/4

- The **protected** modifier specifies that the member can only be accessed within its own package (as with *package-private*) and, in addition, by a subclass of its class in another package.

No-modifier (*package-private*)



The **protected** modifier

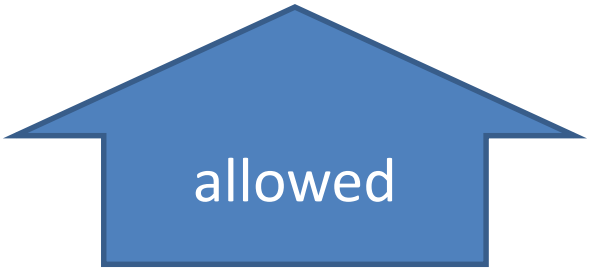


The protected access modifier 2/4

```
1. package com.brainacad.oop1.mycars;
2. public class Car {
3.     protected int maxSpeed = 180;
4.     //...
5.     protected  void move() {
6.         System.out.println(maxSpeed);
7.     }
8. }
```

The protected access modifier 3/4

```
1. package com.brainacad.oop1.spcars2;
2. import com.brainacad.oop1.mycars.Car;
3. public class SportCar extends Car {
4.     //...
5.     @Override
6.     public void move() {
7.         System.out.println( maxSpeed+100);
8.     }
9. }
```



The protected access modifier 4/4

```
1.  package com.brainacad.oop1.test;
2.  import com.brainacad.oop1.mycars.Car;
3.  import com.brainacad.oop1.spcars2.SportCar;
4.  public class Main {
5.      public static void main(String[] arg){
6.          Car myCar1 = new Car();
7.          SportCar myCar2 = new SportCar();
8.          int s1 = myCar1.maxSpeed; //inaccessible
9.          int s2 = myCar2.maxSpeed; //inaccessible
10.         myCar1.move();//inaccessible
11.         myCar2.move();// allowed
12.     }
13. }
```