Utilizarea repetată a claselor.

1. Sintacsa compoziției
2. Sintacsa moștenirii
3. Combinarea compoziției și a moștenirii

Sintcsa compoziției

```
class Source
 { private String s;
 Source()
{ System.out.println("WaterSource()");
 s = new String("Constructed"); }
 public String toString()
 { return s;
 } }
public class SystemA
 { private String valve1, valve2, valve3, valve4;
 Source source;
int i;
 float f;
```

continuare

```
void print()
 { System.out.println("valve1 = " + valve1);
System.out.println("valve2 = " + valve2);
System.out.println("valve3 = " + valve3);
System.out.println("valve4 = " + valve4);
System.out.println("i = " + i);
System.out.println("f = " + f);
System.out.println("source = " + source); }
public static void main(String[] args)
 { SystemA x = new SystemA();
 x.print();
} }
```

Rezultatul realizării:

```
run:
valve1 = null
valve2 = null
valve3 = null
valve4 = null
i = 0
f = 0.0
source = null
```

Inițializarea în constructor prin compoziție

```
class Soap
{ private String s;
 Soap()
 { System.out.println("Soap()");
 s = new String("Constructed"); }
 public String toString()
 { return s; } }
 public class Bath
 {
// Inițializarea la declarare
 private String  s1 = new String("Happy"), s2 = "Happy", s3, s4;
 Soap cast;
 int i;
 float toy;
 Bath()
{ System.out.println("Inside Bath()");
//inițializarea în constructor
 s3 = new String("Joy");
 i = 47;
 toy = 3.14;
 cast = new Soap(); }
```

continuare

```
void print()
 {
// Inițializarea la realizare:
 if(s4 == null)
s4 = new String("Joy");
 System.out.println("s1 = " + s1);
 System.out.println("s2 = " + s2);
 System.out.println("s3 = " + s3);
 System.out.println("s4 = " + s4);
System.out.println("i = " + i);
 System.out.println("toy = " + toy);
System.out.println("cast = " + cast);
 }
 public static void main(String[] args)
 {
Bath b = new Bath();
 b.print();
} }
```

Rezultatul realizării:

```
run:
Inside Bath()
Soap()
s1 = Happy
s2 = Happy
s3 = Joy
s4 = Joy
i = 47
toy = 3.14
cast = Constructed
```

Compoziția cu obiecte public.

```
class Engine
{ public void start() {}
public void rev() {}
public void stop() {} }
 class Wheel
{ public void inflate(int psi) {
System.out.println(”inflate” + psi);
} }
 class Window
 { public void rollup() {
System.out.println(” rollup”);}
 public void rolldown() {} }
class Door
 { public Window window = new Window();
 public void open() {}
 public void close() {} }
```

continuare

```
public class Car
{ public Engine engine = new Engine();
 public Wheel[] wheel = new Wheel[4];
 public Door left = new Door(),
right = new Door(); // 2-door
public Car()
 { for(int i = 0; i < 4; i++)
 wheel[i] = new Wheel(); }
 public static void main(String[] args)
 { Car car = new Car();
car.left.window.rollup();
 car.wheel[0].inflate(72);
 } }
```

Rezultatul realizării:

run:

rollup

inflate72

Sintacsa moștenirii

```
class Cleanser
 { private String s = new String("Cleanser");
public void append(String a)
 { s += a; }
public void dilute()
 { append(" dilute()"); }
 public void apply()
 { append(" apply()"); }
 public void scrub()
{ append(" scrub()"); }
 public void print()
{ System.out.println(s); }
 public static void main(String[] args)
{ Cleanser x = new Cleanser();
 x.dilute();
 x.apply();
x.scrub();
 x.print(); } }
```

continuare

```
public class Detergent extends Cleanser
{ // Suprascrim metoda clasei de bază
public void scrub()
{ append(" Detergent.scrub()");
 super.scrub();
// Activăm metoda clasei de bază }
public void foam()
{ append(" foam()"); }
public static void main(String[] args)
 { Detergent x = new Detergent();
 x.dilute();
 x.apply();
x.scrub();
x.foam();
x.print();
System.out.println("Testing base class:");
Cleanser.main(args); } }
```

continuare

Rezultatul realizarii:

run-single:
Cleanser dilute() apply() Detergent.scrub() scrub()
foam()
Testing base class:
Cleanser dilute() apply() scrub()

Activarea constructorilor fără parametri la moștenire

```
class Art { Art()
 { System.out.println("Art constructor");
 } }
class Drawing extends Art
{ Drawing()
{ System.out.println("Drawing constructor");
 } }
public class Cartoon extends Drawing
{ Cartoon()
 { System.out.println("Cartoon constructor"); }
 public static void main(String[] args)
 { Cartoon x = new Cartoon();
 } }
```

Rezultatul realizării:

```
run:
Art constructor
Drawing constructor
Cartoon constructor
```

Activarea constructorilor cu parametri la moștenire.

```
class Game
{ Game(int i)
 { System.out.println("Game constructor");
} }
class BoardGame extends Game
{ BoardGame(int i)
 { super(i);
 System.out.println("BoardGame constructor");
 } }
 public class Chess extends BoardGame
 { Chess()
 { super(11);
 System.out.println("Chess constructor"); }
 public static void main(String[] args)
 { Chess x = new Chess();
} }
```

continuare

Rezultatul realizării:

run-single:
Game constructor
BoardGame constructor
Chess constructor

Utilizarea compoziției și a moștenirii

```
class Plate
{ Plate(int i)
{ System.out.println("Plate constructor"); } }
class DinnerPlate extends Plate
 { DinnerPlate(int i)
 { super(i);
System.out.println( "DinnerPlate constructor"); } }
 class Utensil
{ Utensil(int i)
{ System.out.println("Utensil constructor"); } }
class Spoon extends Utensil
 { Spoon(int i)
 { super(i);
 System.out.println("Spoon constructor"); } }
 class Fork extends Utensil
 { Fork(int i)
 { super(i);
 System.out.println("Fork constructor"); } }
```

continuare

```
class Knife extends Utensil
{ Knife(int i)
 { super(i);
System.out.println("Knife constructor"); } }
class Custom
{ Custom(int i)
{ System.out.println("Custom constructor"); } }
public class PlaceSetting extends Custom
 { Spoon sp;
 Fork frk;
 Knife kn;
 DinnerPlate pl;
PlaceSetting(int i)
 { super(i + 1);
 sp = new Spoon(i + 2);
 frk = new Fork(i + 3);
kn = new Knife(i + 4);
 pl = new DinnerPlate(i + 5);
System.out.println( "PlaceSetting constructor"); }
 public static void main(String[] args)
 { PlaceSetting x = new PlaceSetting(9); } }
```

Rezultatul realizării:

```
run:
Custom constructor
Utensil constructor
Spoon constructor
Utensil constructor
Fork constructor
Utensil constructor
Knife constructor
Plate constructor
DinnerPlate constructor
PlaceSetting constructor
```

Suprascrierea obiectului clasei de bază în clasa derivată

```
class Homer
 { char doh(char c)
 { System.out.println("doh(char)");
return 'd'; }
float doh(float f)
 { System.out.println("doh(float)");
 return 1.0f; } }
 class Milhouse {}
class Bart extends Homer
{ void doh(Milhouse m) {
System.out.println("Referinta Milhouse");
} }
public class Hide
{ public static void main(String[] args)
 { Bart b = new Bart();
 b.doh(1); // doh(float) folosește
 b.doh('x');
 b.doh(1.0f);
 b.doh(new Milhouse()); } }
```

continuare

Rezultatul realizării:

run:

doh(float)1.0

doh(char)x

doh(float)1.0

Referinta Milhouse

Aducerea la tipul de bază

```
import java.util.*;
 class Instrument
 { public void play() {}
static void tune(Instrument i)
 { // ... i.play();
} }
 // Obiectul Wind este și Instrument
 // deoarece au interfață comună:
class Wind extends Instrument

 { public static void main(String[] args)
{ Wind flute = new Wind();
 Instrument.tune(flute);

} }
```

Inițializarea la moștenire

```
class Insect
 { int i = 9;
 int j;
 Insect()
 { prt("i = " + i + ", j = " + j);
 j = 39; }
 static int x1 = prt("static Insect.x1 initialized");
static int prt(String s)
{ System.out.println(s);
 return 47; } }
 public class Beetle extends Insect
 { int k = prt("Beetle.k initialized");
 Beetle()
{ prt("k = " + k);
 prt("j = " + j); }
static int x2 = prt("static Beetle.x2 initialized");
public static void main(String[] args)
{ prt("Beetle constructor");
 Beetle b = new Beetle(); } }
```

Rezultatul realizării:

static Insect.x1initialized
static Beetle.x2 initialized
Beetle constructor
 i = 9,
 j = 0
Beetle.k initialized
k = 47
 j = 39

Cuvîntul chee final

```
class Value
{ int i = 1; }
 public class FinalData
 { // Может быть константой во время компиляции
 final int i1 = 9;
 static final int VAL_TWO = 99; // Обычная public константы:
public static final int VAL_THREE = 39;
 // Не может быть константой во время компиляции:
 final int i4 = (int)(Math.random()*20);
 static final int i5 = (int)(Math.random()*20);
 Value v1 = new Value();
 final Value v2 = new Value();
 static final Value v3 = new Value();
// Массивы:
 final int[] a = { 1, 2, 3, 4, 5, 6 };
 public void print(String id)
{ System.out.println( id + ": " + "i4 = " + i4 + ", i5 = " + i5);
 }
```

continuare

```
public static void main(String[] args)
 { FinalData fd1 = new FinalData();
 //! fd1.i1++; // Ошибка: значение не может быть изменено
fd1.v2.i++; // Объект не константа!
fd1.v1 = new Value(); // OK -- не final
for(int i = 0; i < fd1.a.length; i++)
fd1.a[i]++; // Объект не константа!
//! fd1.v2 = new Value(); // Ошибка: Нельзя
 //! fd1.v3 = new Value(); // изменить ссылку
//! fd1.a = new int[3];
fd1.print("fd1");
 System.out.println("Creating new FinalData");
 FinalData fd2 = new FinalData();
 fd1.print("fd1");
 fd2.print("fd2");
 } }
```

Parametri final

```
class Gizmo
 { public void spin() {} }
 public class FinalArguments
{ void with(final Gizmo g)
 { //! g = new Gizmo(); // Неверно -- g - final }
 void without(Gizmo g)
{ g = new Gizmo(); // OK -- g не final
 g.spin(); }
 // void f(final int i) { i++; } // Не может измениться // Вы можете
     только читать примитив:
 int g(final int i)
 { return i + 1; }
 public static void main(String[] args)
{ FinalArguments bf = new FinalArguments(); bf.without(null);
 bf.with(null);
} }
```

continuare

Rezultatul realizării:

run-single:
fd1: i4 = 6, i5 = 18
Creating new FinalData
fd1: i4 = 6, i5 = 18
fd2: i4 = 15, i5 = 18

Clase final

```
class SmallBrain {}
final class Dinosaur
{ int i = 7;
 int j = 1;
SmallBrain x = new SmallBrain();
 void f() {} }
//! класс Further расширяет Dinosaur {}
 // Eroare: clasa 'Dinosaur' nu poate fi moștenită
public class Jurassic
{ public static void main(String[] args)
{ Dinosaur n = new Dinosaur();
 n.f();
 n.i = 40;
 n.j++;
 } }
```