Programazio modularra Java-n

Programazioa II

7. gaia

Helburu nagusiak

» Java lengoaiaren ezaugarri nagusiak aurkeztea, OOParen aldetik batik bat.

» Javan idatzitako programen egitura azaltzea.

Aurkezpenaren eskema

- Sarrera gisa
  - » Historia pixka bat. Java, Interneterako lengoaia.
  - » Java ingurunea: programazio-lengoaia, JVM, eta Java-ko paketeak.
  - » Lengoaiaren ezaugarri orokorrak.
- Java, OOP lengoaia
  - » Klaseak, ikusgaitasuna (modifikatzaileak), eremuak eta metodoak, herentzia, polimorfismoa, klase eta metodo abstraktuak, eta interfazeak.
  - » Programen antolaketa orokorra.
  - » Java *vs.* Ada.
- Bukatzeko
  - » Erabiltzaile interfaze grafikoak (GUI), J2SE (lehen JDK), eta helbide interesgarriak.

Historia

- Sun Microsystems enpresak garatua.
- Hastapenean, etxetresna elektrikoak programatzeko.
- Amarauna hedatu ahala, Java indartuz doa web-eko aplikazioetarako.
- 1995: Netscape nabigatzaileak Java sostengatzen du. Hortik aurrera Interneterako lengoaiatzat hartu ohi da.

Java, Interneterako lengoaia

- Amarauneko edukiaren pasibotasun eta estatikotasun gabeziari erantzuten dio.
- Aplikazioak eta *applet*-ak
  - » Java aplikazioa: arrunta, bere kasa dabilena.
  - » *Applet-a*: HTML dokumentuaren “kargarekin” batera exekutatzen dena.

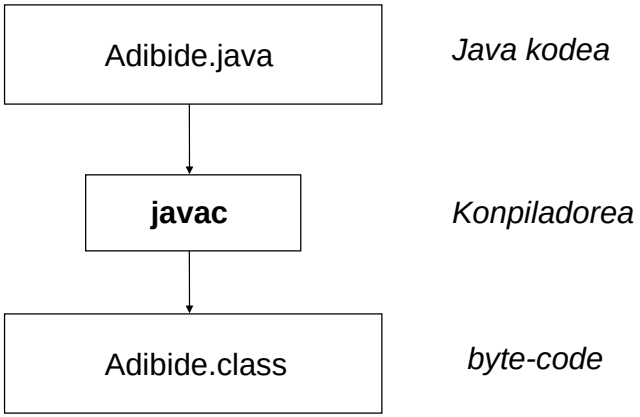
Java ingurunea

- *Java inguruneak* hiru oinarrizko elementu ditu:
  - » Java programazio-lengoaia.
  - » Java makina birtuala (JVM).
  - » Java klaseen multzoa.

Java, programazio-lengoaia

- Java helburu orokorreko lengoaia da.
  - » Objektuei orientatua.
  - » C eta C++ lengoaien ahaidea.

Konpilazioa



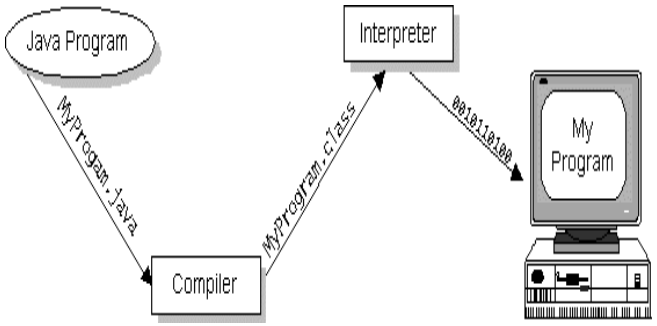
Java "makina birtuala" (*Java Virtual Machine*)

- *Java "makina birtuala" (JVM)*
  - » Edozein makinatan Java programak egin litezke, eta gero beste edozeinetan-edo exekutatu.

Java, JVMrako lengoaia

- Makina horretarako programei byte-code deitzen zaie.
- Javarekin batera garatutako lengoaia.
  - » Gehienez 256 agindu. Gaur egun 220 aginduz osatua.
- Beste lengoaia batzuk ere konpila daitezke JVMrako: *jgnat* (Ada -> JVM).

Bi fase: konpilazioa eta interpretazioa



Java paketeak (klase-bildumak)

- Paketea: elkarrekin zerikusi estua duten klaseen multzoa.
- Pakete arruntak:
  - » `java.lang`
  - » `java.io`
  - » `java.util`
  - » `java.net`
  - » `java.awt`
  - » `java.applet`

Lengoaiaren ezaugarri orokorrak (I)

- Sinplea
- Objektuei zuzendua
- Banatua
  - » TCP/IP konexioetarako erraztasunak. Kode-zatiak nonahi (banatuta) egon litezke.
- Segurua
  - » Mota-mekanismo zorrotza (ez Adarena bezain zorrotza, halere).
  - » Konpilazio-denborako egiaztapena (*checking*).
  - » Interpretazio/exekuzio denborako egiaztapena (*checking*).
  - » Erakusle espliziturik ez.
  - » Salbuespenen tratamendua.

Lengoaiaren ezaugarri orokorrak (II)

- Segurua (erasoen aurrean)
  - » Lengoaiak berak babesten du memoria atzipen “ilegalen” aurrean.
- Eramangarria
  - » Edozein plataformatara eramateko espezifikatua.
- “Exekuzio-hari” anitzekoa (multithreaded)
  - » Prozesuen egikaritzapen konkurrenterako aukera ematen du.
- Interpretatua
- Dinamikoa
  - » Beharra sortu ahala kargatzen ditu klaseak interpretatzaileak.

Lengoaia: oinarrizko datu-motak

- Datu-motak:
  - » Osokoak: `byte, short, int, long`
  - » Errealak: `float, double`
  - » Karaktereak: `char`
  - » Boolearrak: `boolean`
- Array-ak (C-koen antzera); bestalde, `Vector` klasea ere badago.
- `String` klasea.

Lengoaia: elementuak (I)

- *C* eta *C++* lengoaien itxura:
  - » Iruzkinak.
  - » Identifikadoreak.
  - » Literalak.
  - » Eragileak.
  - » Banatzaileak.

Lengoaia: elementuak (II)

- Fluxu-kontrola:
  - » Baldintzazkoak.
  - » Begiztak.
  - » Salbuespenak.
  - » Fluxuaren kontrol orokorra.
- Sententzia konposatua (sekuentzia):
  - » Giltza artean {}

Lengoaia: elementuak (III)

- Baldintza-sententziak:
  - » **`if`** … **`if … else`**
  - » **`switch`**
- Iterazioa:
  - » **`while`** **(**<espresio boolearra>**)**
    <sententzia bakuna edo konposatua>
  - » **`do`**
    <sententzia bakuna edo konposatua>
    `while` **(**< espresio boolearra >**)**
  - » **`for`** **(**<hasieratze-sententziak>;
    <espresio boolearra>;
    <aldatze-espresioak>**)**
    <sententzia bakuna edo konposatua>

Lengoaia: elementuak (IV)

- Adibide bat (`for`):

```
for (int i=0; i<5; i++)
  System.out.println ("Kaixo");
for (int i=0; i<5; i++){
  System.out.println ("Kaixo");
//…;
}
```


Lengoaia: salbuespenak (I)

- Javak baditu salbuespenak.
- Programa baten exekuzioan errorea gertatutakoan, errorea detektatzen duen kodeak *salbuespena sorrarazten du* eta, gero, errorea *harrapatu* (tratatu) egiten da.
- Salbuespenak `Exception` klasekoak edo azpiklaseren batekoak dira.

Lengoaia: salbuespenak (II)

- Aurredefinitutako salbuespen asko eskaintzen du:
  - `java.io.EOFException, java.io.FileNotFoundException`

  (sarrera/irteerakoak, adibidez)
- Edo exekuzio-denborako beste hauek:
  - `ArithmeticException`
  - `NumberFormatException`
  - `IndexOutOfBoundsException`
  - `NegativeArraySize`
  - `NullPointerException`

Lengoaia: salbuespenak (III)

try-catch-throw:

```
try {
  sententziak;
[throw (salbuespena);]
 sententziak;

} catch (salbuespena) {
    sententziak;
}
```


Lengoaia: salbuespenak (IV)

- Adibide bat:

```
public void writeList() throws IOException,
  ArrayIndexOutOfBoundsException {...

try { //…;
      writeList();
      //…; }
catch (ArrayIndexOutOfBoundsException e) {
  System.err.println("Array-aren mugak gainditua: "
    + e.getMessage()); }
catch (IOException e) {
  System.err.println("S/I-ko errorea: " +
    e.getMessage()); }
```


Modulartasuna Javan: kapsulaketa eta informazio-ezkutatzea

- Klaseak (Ada-ko paketeen kidekoak)
- Informazio-ezkutatzea (ikusgaitasuna, oro har) modifikatzaileen bidez gauzatzen da: `private`, `protected`, `public`...

Java, objektuei zuzenduriko programazio-lengoaia

- Klaseak.
  - » Eremuak: oinarrizko motetako aldagaiak eta objektu-aldagaiak
  - » Metodoak: eragiketak.
- Ikusgaitasuna eta atzipen-kontrola (modifikatzaileak):
  `protected, public, private, ...`
- Herentzia: `Object` klase gorena.
- Gainkarga.
- Klase abstraktuak.

Klaseak

- Klaseak dira, Java-n, OOParen oinarria.
- Java programa: elkarrekin erlazionaturiko klase bat edo gehiago.
- Aldagai eta funtzio oro klase baten baitan erazagutu beharra dago.
- Programa bat exekutatzerakoan, klase nagusiaren izena ematen da eta bertan `main` izeneko metodoa bilatzen da (horixe da exekuzioari hasiera ematen diona).

<klase-definizioa> ::=
[**public**] [**abstract**] [**final**] **class** <klase-izena> **{**
<klaseko-kidearen-erazagupena> *
}

Modulartasuna Javan: klasea

- Klasea = Javako modulu-unitate nagusia
  - DMA bat edo EMA bat inplementatzeko egokia
  - Informazio-ezkutatzea gauzatzeko egokia
  - Objektu edo instantzia multzo baten abstrakzioa errepresentatzen du
- Programa bat hainbat klase erlazionatuk osatzen dute (gutxienez batek).
  - Aldagai oro eta eragiketa oro klase baten barruan erazagutu edo/eta inplementatu behar da.
  - Java programa baten exekuzioa bere klase nagusitik hasten da, bertako `main` metodoa exekutatuz.

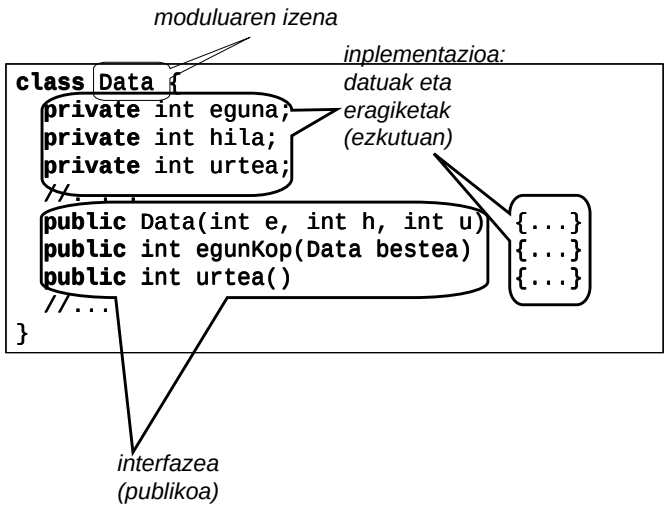
Modulartasuna Javan: klasea

Data klasearen atributuak edo **eremuak**:
- eguna,
- hila
- urtea.

Datak maneiatzeko, eragiketa edo **metodo** batzuk behar dira:
- klaseko objektuak eraiki (instantziatu)
- bi dataren arteko egun kopurua kalkulatu
- data baten urtea zein den jakin
- ...

```
class Data {
  private int eguna;
  private int hila;
  private int urtea;
  //. . .
  public Data(int e, int h, int u)
    {...}
  public int egunKop(Data bestea)
    {...}
  public int urtea()
    {...}
  //...
}
```

Modulartasuna Javan: klasea

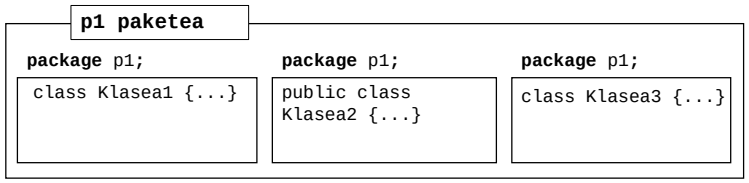


Modulartasuna Javan: klasea

- Klaseak
  - » Eremuak: oinarrizko motetako aldagaiak eta objektu- aldagaiak (edo erreferentzia-aldagaiak).
  - » Metodoak: eragiketak.
- Ikusgaitasuna eta atzipen-kontrola (modifikatzaileak): `protected, public, private...`
- Objektuak: klaseen instantzia.

Ikusgaitasuna (I)

- Klase bat, berez, izengabeko paketekoa da (pakete lehenetsia, besterik ezekoa).
- Aplikazioko klaseak paketetan antolatu nahi badira, iturburu-programak paketeen izen bereko direktorioetan kokatu behar dira, pakete-egitura islatzen duen direktorio-egitura bat eratuz (hori dena, gehienetan, programazio-inguruneak eginda emango digu).
- Klase bat pakete batekoa dela adierazteko, **`package`** klausula erabili behar da (klasearen aurretik).
- Java paketeak: zentzu batean, Ada-z pakete umeak erabiliz eratutako pakete-hierarkien antzekoak dira.



Ikusgaitasuna (II)

- **`public`**: klase guztietarik ikusgai (irudian, Klasea2 eta Klasea3).
- *package* (besterik ezean): **`public`** modifikatzailerik ez badago, ikusgaitasunari *package* esaten zaio: klasea bere paketeko klaseetatik bakarrik ikusiko da (irudian, Klasea1).

Ikusgaitasuna (III)

- Beste paketeetako **`public`** klaseak ikusgai dira, beti ere beren paketearen izena aurretik jarriz kalifikatuz gero.
- Klase-izen soila erabili nahi izanez gero, **`import`** klausula erabili behar da (Ada-ko `use`-ren antzera).

  `import` <pakete-izena> [{**`*`** | <klase-izena>}] **`;`**

  `*`: paketeko klase guztiak inportatzen dira

  <klase-izena>: aipatutako klasea soilik inportatzen da

Klaseen definizioa: kideak (I)

- Klase baten ezaugarriak eta funtzionalitatea eremu eta metodoen bitartez definitzen dira (edozein ordenatan erazagutuz): klaseko kideak (*members*) dira.
- Eremuak: oinarrizko motetako aldagaiak edo objektu-erreferentziak (klaseren bateko instantziak).

Klaseen definizioa: kideak (II)

```
class MotorKlase { . . . }
class Kamioi {
  private int gurpilKopurua;
       // aldagai osoa; pribatua
  private double pisua;
       // aldagai erreala; pribatua
  protected MotorKlase motorra ;
       // objektu-aldagaia; babestua
  public char[] matrikula;
       // karakterezko array-a; publikoa
  ...
  }
```


Kideen modifikatzaileak (I)

<modifikatzailea>::=

<atzipenaren-zehaztapena>: atzipena zehazten du. Ikusgaitasuna (kapsulaketa).

<partekotzaren-definizioa>: eremua edota metodoa instantzia guztiek daukaten ala klasearen ezaugarri berezko eta bakarra den zehazten du.

<kide-konstantea>: balioa alda litekeen ala ez adierazten du.

Kideen modifikatzaileak (II)

<atzipenaren-zehaztapena>::=

`public`: aldagaia edo metodoa edozein klasetatik ikusgai dago.

`protected`: aldagaia edo metodoa bere klasetik, klase umeetatik, eta bere paketeko klase guztietatik dago ikusgai.

`private`: aldagaia edo metodoa bere klasean bakarrik dago ikusgai.

package (besterik ezean): aldagaia edo metodoa bere paketeko edozein klasetatik dago ikusgai.

Kideen modifikatzaileak (III)

<partekotzaren-definizioa>::=

`static`: aldagaia edo metodoa klaseari dagokio soilik (ez da heredatzen). Klaseko aldagai edo metodo esaten zaie.

[ ] (ez bada zehazten, **ez** dira **estatikoak**): objektu (instantzia) bakoitzak aldagaiaren edo metodoaren bere bertsioa izango du (heredatu egingo du).

- Klase bateko aldagai edo metodo estatikoak leku bakar batean gordetzen dira; ez-estatikoak, berriz, dagokion objektuan gordetzen dira.
- Objektu bat instantziatzean (dinamikoki sortzean) aldagai/metodo ez-estatiko bakoitzaren kopia bat sortzen da.

Adibidea: **`static`**

```
public class K {
    private static int zenbatInstantzia=0;
    private char c; //eremu hau instantziek edukiko dute
    public K(){
        c='';
        zenbatInstantzia++;
    }
    public static int zenbatAle() {return zenbatInstantzia;}
}

public class Proba {
    public static void main(String[] args) {
        K k1=new K();
        K k2=new K();
        System.out.println (K.zenbatAle());
    }
}
```


Kideen modifikatzaileak (IV)

<kide-konstantea>::=

`final`:

eremua: konstante bat da.

metodoa: ezin da azpiklaseetan birdefinitu.

[ ] (ez bada zehazten, **ez** da **finala**): ez bada finala, balioa aldatzen ahal zaio.

- Konstante bat klase mailan baino ezin da definitu (ezin da, beraz, metodoetan).

Informazioaren ezkutatzea: ikusgaitasuna (laburpen gisa)

» Klaseak publiko erazagut daitezke (`public`) edo *package* ikusgaitasun-mailarekin (pakete bereko beste klaseetatik baino ikusten ez direla).

» Aldagai kideak (klaseko aldagaiak) eta metodoak (eragiketak) `public, private, protected` edo *package* izan daitezke.

Eremuak: aldagaiak eta objektuak (I)

Eremua: oinarrizko mota bateko aldagaia edo objektu baten erreferentzia.

<eremu-erazag.> ::= {<mota-izena> | <klase-izena>}
<eremu-ident.> [= <espresioa>]**;**

<mota-izena>: aldagaia (datu-eremua)

<klase-izena>: klase horretako objektu baten erreferentzia (instantzia)

Datu-eremua (aldagaia)

- Aldagai oro klase baten baitan erazagutu behar da.
- Besterik ezeko hasieratzea: boolearrak, `false`; `char`, `'\0'`; ...).
- Aldagaiak ikusgai daude erazagutu diren blokean.

Eremuak: aldagaiak eta objektuak (II)

Objektu-eremua

- Eremu bat objektu baten erreferentzia izan dadin nahi bada, objektua sortu beharko da `new` eragilearen bitartez (instantziazioa):

<klase-izena> <eremu-ident.> =
new <eraikitzailearen-izena> [<parametroak>]*

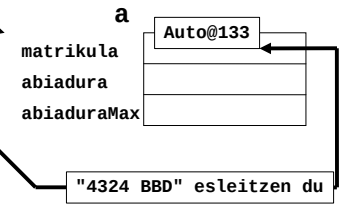
- Sorkuntzan automatikoki deitzen zaio klaseko eraikitzaileari.
- Horrela lortzen da, sortu berri den erreferentzia objektu-aldagai horrek izan dezan.
- Klase horretan erazaguturiko eremu ez-estatiko guztien kopia bat sortzen da instantzian.

Eremuak: aldagaiak eta objektuak (III)

```
Auto a = new Auto();


a.matrikula = "4324 BBD";
a.abiadura = 70.0;
a.abiaduraMax = 123.45;

System.out.println
  (a.matrikula + " autoa " + a.abiadura +
   " abiaduran doa.");
```



Puntu-notazioa erabiltzen da, klase bateko ezaugarriak atzitzeko

Metodoak (I)

- Klase bateko metodoek klase horretako objektuek egin ditzaketen eragiketak (funtzionalitatea) definitzen dituzte.
- Metodoa: sentenzia-sekuentzia bat.
- Parametroak: metodoaren argumentuak.
- Parametro-pasea: balioz beti (kopia).
- Metodoen barruan ere erazagut daitezke barne-klaseak (habiratuak).

<metodo-erazag.> ::= {**`void`** | <mota-izena> | <klase-izena>}
<metodo-ident> **(** [<param.-lista>] **) {**
<sententziak>
}

`void`: "prozedura" dela adierazten du, ez baitu inongo baliorik itzultzen emaitza gisa (bestela, "funtzioa" da).

Metodoak (II)

- **`main`**: metodo berezia, JVMtik bere klaseari dei egiten zaionean exekutatzen dena.
- Eraikitzaileak: klasearen izen bera duten metodo bereziak.
- Klasea sortzen duen programatzaileak eraikitzaile bat edo gehiago idatz ditzake (parametro desberdinekin), eta horiek izango dira objektuak sortzean erabiliko direnak.

<eraikitzailearen-erazag.> ::=
<klase-ident.> **(** [<param.-lista>] **) {**
<sententzia-sekuentzia>
}

Metodoak (III)

- Metodoa "funtzioa" denean, `return` sententzia bat izan behar du bere gorputzean.
- Metodo bati deitzeko, dagokion objektuaren izena jarriko zaio aurretik (objektu horrek "jasotzen du mezua"):

<objektuaren-izena>**.**<bere-klaseko-metodoaren-izena>

- Klaseko metodoak: **`static`** modifikatzailearekin erazagutuak, klasearen izena aurretik jarriz deitzen zaie.

Metodoak (IV)

```
class Auto {
  String matrikula;
  double abiadura;
  double abiaduraMax;
  // bizkortu topera
  void bizkortuMax() {
    this.abiadura = this.abiaduraMax;
  } }
```

`this` uneko objektua da, hau da, `bizkortuMax` eragiketa egiten/jasaten ari dena

Hala ere, kasu honetan ez da beharrezkoa **`this`** erabiltzea:

```
abiadura=abiaduraMax;
```

Klasearen ezaugarriek, klasekoak **nolakoak diren** errepresentatzen dute**.**

Metodoek, klaseko objektuek zer **egiten/jasaten duten** adierazten dute (portaera)**.**

Metodoak (V)

```
Auto a = new Auto();

a.matrikula = ”SS-4324-BD";
a.abiadura = 70.0;
a.abiaduraMax= 123.45;

System.out.println(a.matrikula + " autoa " + a.abiadura +
   " abiaduran doa.");
a.bizkortuMax();

System.out.println(a.matrikula + " autoa " + a.abiadura +
  " abiaduran doa.");
```

Objektuek, **egoera edo balioa** mantentzen dute une oro.

> Metodoak ere puntu (.) eragilearen bidez atzitzen dira.

Metodoak (VI): gainkarga

```
class Puntu {
  private double x;
  private double y;
  void print() {
    System.out.println("Puntua: " + this.x + "," + this.y);}
  void print(int n) {
    for (int i = 0; i < n; i++) {
      System.out.println("(" + this.x + "," + this.y + ")");
      }
  }
}
```

`Puntu` klaseko **`print()`** eta **`print(n)`** metodoak desberdinak dira: **`print`** gainkargatuta (overloaded) dagoela esaten da.

DMA-a: adibidea Javaz

```
/**
 * Ikasle DMA-a inplementatzen duen klasea
 */
public class Ikasle {
   // klasearen barne-errepresentazioa: eremuak
   private String izena; //ikaslearen izena
   private Talde taldea; //ikasleari dagokion taldea
   private Nota nota; //ikaslearen nota

   /**
    * Eraikitzailea: ikasle bat sortzen du, izena eta taldea emanik
    * @param izena ikaslearen izena
    * @param taldea ikasleari dagokion taldea
    */
   public Ikasle(String izena, Talde taldea) {
    this.izena = izena;
    this.taldea = taldea;
   }
```


DMA-a: adibidea Javaz

```
   // kontsulta-metodoak
   public String izena() {
    return this.izena;
   }
   public Talde taldea() {
    return this.taldea;
   }
   public Nota nota() {
    return this.nota;
   }
   // aldaketa-metodoak
   public void aldatuTaldea (Talde t) {
    this.taldea = t;
   }
   public void esleituNota (Nota n) {
    this.nota = n;
   }
}
```

DMA-a: adibidea Javaz

```
/**
 * IkasleProba: Ikasle klasea probatzeko programa
 */
public class IkasleProba {
  public static void main(String[] args) {
    Ikasle i1, i2;
    i1 = new Ikasle("Miriam", new Talde("SIIT46"));
    i2 = new Ikasle("Xabier", new Talde("SIIT46"));
    //...
    i1.esleituNota(new Nota(4.5));
    System.out.print(i1.izena() + " ikaslearen nota: ");
    i1.nota().put();
    System.out.println();
    //...
  }
}
```

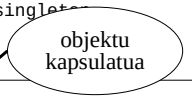
EMA-ak Javaz: *singleton* patroia

- DMAek ez bezala, EMA baten eragiketak objektu bakar baten gainean egiten dira. Ezin dira klase horretako instantziak egin.
- *Singleton* patroia erabiliz idazten dira EMA-ak Javaz.
- *Singleton* patroia:
  - Eraikitzaileak pribatua izan behar du (klasearen instantziak egitea eragozteko)
  - Atributuak klase mailakoak izango dira (`static`)
  - Metodoek klasearen barruko objektu kapsulatuaren gainean izango dute eragina, eta hauek ere estatikoak izango dira

EMA-a (*singleton*): adibidea Javaz

```
/**
 * Erasmusekoen ikasgela, objektu-abstrakzioa (EMA-a, singleton
   patroia)
 */
public class ErasmusIkasgela {
   // konstanteak
   private static final int KOP_MAX = 50;
   private static final String FITXATEGI_IZENA = "erasmus.txt";

   // objektu kapsulatua(k)
   // obj. kapsulatu nagusia: ikasleen bektorea
   private static Ikasle[] ikasgela = new Ikasle[KOP_MAX];
   // azken ikaslearen kokapena:
   private static int azkena = -1;
```



EMA-a *(singleton):* adibidea Javaz

```
/**
     * eraikitzailea (pribatua, hutsik):
     * inork ezin du klase honen instantziarik eraiki
     */
    private ErasmusIkasgela(){
    }

    // hasieratzea
    /**
     * eratu ikasgela, datuak fitxategitik irakurriz;
     * ordenatu array-a, notaren arabera (adibidez)
     */
    public static void kargatuFitxategitik() {//inplementatzeke dago
    }
```

EMA-a *(singleton):* adibidea Javaz

```
    // kontsulta
    public static Ikasle ikaslerikOnena() {
     return ikasgela[0]; // ordenatuta dagoenez, lehena itzultzen du
    }
    public static int zenbatIkasle() {
     return azkena + 1;
    }
    // aldaketa
    public static void ikasleaGehitu(Ikasle i) {
     ikasgela[azkena + 1] = i;
     azkena =+ 1;
    }
    public static void ordenatu() { //inplementatzeke dago
    }
}
```


EMA-a *(singleton):* adibidea Javaz

```
public class ErasmusIkasgelaProba {
  public static void main(String[] args) {
    Ikasle i1;
    ErasmusIkasgela.kargatuFitxategitik();
    // ikasle berri bat sortu, eta nota esleitu
    i1 = new Ikasle("Miriam", new Talde("SIIT"));
    i1.esleituNota(new Nota(8.5));
    // ikaslea ikasgelan sartu eta ikasgela ordenatu
    ErasmusIkasgela.ikasleaGehitu(i1);
    ErasmusIkasgela.ordenatu();
    // ikasgelako onenaren izan inprimatu
    System.out.println ("Ikasle onena: " +
      ErasmusIkasgela.ikaslerikOnena().izena());
  }
}
```

metodoak klasearenak dira, ez objektu edo instantzia batenak

Herentzia

- Klase bat beste batetik eratorrarazten baldin badugu, haren ezaugarriak (aldagaiak eta metodoak) heredatuko ditu (kontrakoa ez bada zehazten, noski).

  <klase-definizioa> ::=
  [**public**] **class** <klase-izena>
  [**extends** <existitzen-den-klase-baten-izena>]**{**
  <klaseko-kidearen-erazag.> *
  }

- Klase eratorriak aldagai eta metodo berriak gehi ditzake (espezializazioa) eta/edo heredaturiko metodoak birdefinitu.
- Bada klase erro lehenetsi bat, **`Object`**, zeinen eratorpena diren beste guztiak. Klase-hierarkiak era daitezke honela.

Adibidea: klase-hedapena (I)

```
class BankuKontu {
  protected double vSaldoa;
  // aldagaia
  public double saldoa()
  {            return
  vSaldoa;
      } // metodoa
}
class NireKlasea {
  public static void sartuDirua() {
        KontuKorronte cc =  new KontuKorronte();
        cc.sartu(125000.00); // kontu korrontean sartu
        System.out.println("Kontu korrontearen saldoa =" +
    cc.saldoa());          // saldoa inprimatu
  }
}
```

```
class KontuKorronte extends
  BankuKontu {
  public void sartu
    (double kopurua) {
   …
  }
  …
}
```

Adibidea: klase-hedapena (II)

- `KontuKorronte`-ak `BankuKontu`-ak dira *(is-a)*.
- Beraz, `vSaldoa` aldagaia eta `saldoa` metodoa heredatuko ditu.
- Gainera, `sartu` metodo berria gehitzen dio klase honi (espezializazioa).
- `cc` objektuak, `KontuKorronte` klaseko instantzia bat denez, bertako metodo eta aldagaiak atzi ditzake.

Herentzia eta polimorfismoa

- **Polimorfismoa**
  - » Izen bereko metodo desberdinak idatz daitezke hierarkia bateko hainbat klase eratorritan (birdefinizioa, *overriding*).
  - » Exekuzio-denboran erabakiko da zein den exekutatzen dena (*dispatching*).
  - » Objektuari dagokion metodoa da exekutatuko dena.

Adibidea: polimorfismoa (I)

```
class BankuKontu {
  protected double
  vSaldoa;
  public double saldoa() {
        return vSaldoa;
  }
  public void
  interesaMetatu() {
        vSaldoa += vSaldoa
  * 0.10 ;
  }
}
```

```
class KontuKorronte
  extends BankuKontu{
  public void sartu
  (double kopurua) { …
        }
  public void
  interesaMetatu() {
        if (vSaldoa >250000)
         { vSaldoa += vSaldoa
           * 0.15; }
        else {vSaldoa +=
           vSaldoa * 0.10;}
   }
    … //KontuKorronte-ren
      //metodo esklusiboak
}
```


Adibidea: polimorfismoa (II)

```
public class NireKlasea {
  public static void funtzioa (BankuKontu kontua)
  {
   kontua.interesaMetatu();
  }
  public static void BesteFuntzioBat () {
   BankuKontu cb = new BankuKontu();
   funtzioa(cb);   // dei zuzena (egokia)
   KontuKorronte cc = new KontuKorronte();
   funtzioa(cc);   // hau ere zuzena da
  }
}
```

Adibidea: polimorfismoa (III)

- `funtzioa`-ren lehen deia zuzena (egokia) da: `BankuKontu`-ko `interesaMetatu` exekutatuko da.
- Bigarren deia ere egokia da, nahiz eta parametroaren mota estatikoa `BankuKontu` izan eta exekuziokoa (dinamikoa) `KontuKorronte`.
- Kasu hauetan, Java exekuzioko motan oinarritzen da (*dispatching*-a). Horrela, `kontua` parametro formala `BankuKontu` motakoa izanik ere, `funtzioa`-ri `cc` parametroarekin deitzean `KontuKorronte`-ri dagokion `interesaMetatu` exekutatuko da.

Polimorfismoa (IV)

- Datu-mota estatikoa *vs.* datu-mota dinamikoa
  - » Entitate baten datu-mota estatikoa erazagupenean zehazten da.
  - » Exekuzio-garaian, entitatea beste motako objektu batekin lotzen bada, mota hori izango da bere datu-mota dinamikoa.

```
Poligono p = new Poligono();
      p-ren datu-mota estatikoa: Poligono
      p-ren datu-mota dinamikoa: Poligono

Laukizuzen  r = new Laukizuzen ();
p = r; // emanik Poligono Laukizuzen klasearen umea dela
      p-ren datu-mota estatikoa: Poligono
      p-ren datu-mota dinamikoa: Laukizuzen
```


Adibidea: goiko klaseko metodoen deia (I)

```
class BankuKontu {
   protected double vSaldoa;
   public double saldoa() { return vSaldoa; }
   public void interesaMetatu() { vSaldoa += vSaldoa *
   0.10 ; }
}
class KontuKorronte extends BankuKontu {
  public void sartu (double kopurua) { …    }
  public void interesaMetatu() {
        if  (vSaldoa > 250000) { vSaldoa += vSaldoa *
  0.15; }
        else  {// BankuKontu-koari deituz
               super.interesaMetatu(); }
  }
  …            //KontuKorronte-ren metodo esklusiboak
}
```


Klase abstraktuak

- Klase abstraktu batek datu-eremu eta metodo zenbait erazagutzen ditu, baina metodoak definitu gabe (inplementaziorik gabe) utz ditzake.
- Ezin dira klaseko instantziak sortu.
- Abstraktuak klase berriak eratortzeko oinarri izango dira: klase eratorriek klase abstraktuan erazaguturiko metodo abstraktuen gorputzak inplementatu behar dituzte.

<klase-abstraktuaren-def.> ::= **abstract** <klase-def.>
<metodo_abstraktuaren-erazag.> ::= **abstract**
{**void** | <mota-izena> | <klase-izena>}
<metodo-ident.> **(** [<param.-lista>] **);**

Adibidea: klase eta metodo abstraktuak (I)

```
public abstract class BankuKontu
   {   ...
   public abstract void
   interesaMetatu () ;
   ...}
class KontuKorronte extends
   BankuKontu {
   public void interesaMetatu ()
     {…}
  ...}
class AurrezkiKontu extends
   BankuKontu {
   public void interesaMetatu()
     {…}
...}
```

Ezin da egin:

```
BankuKontu bk = new
   BankuKontu();
```

Egin daiteke:

```
KontuKorronte cc = new
   KontuKorronte();

AurrezkiKontu ak = new
   AurrezkiKontu();

BankuKontu bk = new
   AurrezkiKontu();

bk.interesaMetatu();

//AurrezkiKontu.interesaMetatu
```


Adibidea: klase eta metodo abstraktuak (II)

- `interesaMetatu` **abstraktu** egiten dugu, kontu-mota guztiek beren `interesaMetatu` izango dutelako (desberdinak izan daitezke).

Adibidea: datu-egitura heterogeneo baten prozesamendua (I)

```
public class NireKlasea {
//kontu guztietako interesak batzen ditugu;
//kontuak BankuKontu motako array batean gordetzen
  ditugu

  public static void interesaKalkulatu
     (BankuKontu[] bk, int osagaiKop) {
   int intMetatua = 0;
   for ( int i = 0; i < osagaiKop; i++) {
     intMetatua += bk[i].interesaMetatu();
    //eragiketa polimorfikoa: dispatching-a
   }
  }
}
```


Adibidea: datu-egitura heterogeneo baten prozesamendua (II)

- Taulako osagaia (`BankuKontu` motakoa) `KontuKorronte` motakoa denean (exekuzioan), `KontuKorronte`-ko `InteresaMetatu` exekutatuko da.
- Taulako osagaia (`BankuKontu` motakoa) `AurrezkiKontu` motakoa denean, `AurrezkiKontu`-ko `InteresaMetatu` exekutatuko da.

Interfazeak (I)

- **Askotariko herentzia simulatzen** dute, non klase bat, bi gainklase edo gehiagotatik eratortzen den.
- Interfaze bat klase abstraktu berezi bat baino ez da, non erazagut baitaitezke hainbat metodo (emaitzaren mota, izena eta argumentuak), baina definitu (inplementatu) gabe utziz.

`class` Klasea **`implements`** Interfazea

Interfazeak (II)

- Klase bat beste batetik baino ezin da eratorri (`extends`), baina interfaze bat edo gehiagoren inplementazioa (`implements`) izan daiteke. Hala bada, klaseak inplementatzen dituen interfazeek erazaguturiko metodo guztien definizioa (inplementazioa) eman behar du.

<interfaze-def.> ::= [**`public`**] **`interface`** <interfaze-izena> **{**
<metodo-erazag.> *
}
<klase-def.> ::= [**`public`**] **`class`** <klase-izena>
[**`extends`** <existitzen-den-klase-baten-izena>]
[**`implements`** <interfaze-izena>*] **{**
<klaseko-kide-erazag.> *
}

Adibidea: interfazeak

```
public interface Konparagarri {
  public boolean txikiago (Konparagarri bestea);
  public boolean berdin (Konparagarri bestea); }

public class Oso implements Konparagarri {
  private int balioa;
  public Oso (int val) {balioa = val;} //eraik.

  public int balioaIrakur () {return balioa;}

  public boolean berdin (Konparagarri bestea) {
     return this.balioa == ((Oso)bestea).balioa; }

  public boolean txikiago (Konparagarri bestea) {
     return this.balioa < ((Oso)bestea).balioa; }
}
```


Java programen antolaketa (I)

- Diseinua: identifikatu klaseak, eta hierarkikoki sailkatu
- Klaseen egitura

```
public class Klase1 extends GoiKlasea {
//klaseko kideak (members) definitu:
 //aurrena aldagaiak, gero metodoak

 //aldagaiak
  private String izena;
  private int adina;

 //metodoak
  //eraikitzailea
  public Klase1(String izenBat, int adinBat) {
    izena = izenBat;
    adina = adinBat
  }
  public boolean bozkaDezake () {
    return adina >= 18;
  }
}
```

Java programen antolaketa (II)

- Klaseetatik objektuetara (eraikuntza): new
  `Klase1 hura = new Klase1("Anabel", 29);`
- Estiloa
  - » Aldagaiak eta metodoak izendatzeko, puntu-notazioa. Adib.: `hura.bozkaDezake()` edo `this.adina`
    - – Klaseak, gehienetan, maiuskulaz hasten dira
    - – Metodoak, paketeak eta aldagaiak, gehienetan, minuskulaz
- Bestelakoak:
  - » `this`, objektua bera, eta `super`, goiko klasea
  - » Metodoen gain-idazketa (*overriding*)
  - » Metodoen gainkarga (*overloading*).

Java programen antolaketa (III)

- DMA-ek ez bezala, EMA baten eragiketak objektu bakar baten gainean egiten dira. Ezin dira klase horretako instantziak egin.
- *Singleton* patroia erabiliz lortzen dira EMA-ak.
- *Singleton* patroia erabiltzea, ezaugarri hauek betetzen dituen klasea egitea da:
  - » Eraikitzaile pribatua izan beharko du (klasearen instantziak egitea eragozteko)
  - » Atributuak klase mailakoak izango dira (`static`)
  - » Metodoak klasearen barruko objektu bakarraren gainekoak izango dira (`static`)

EMA adibidea: pila

```
public class PilaEmaArray extends Object{

// Atributuak:
 private static final int EDUKIERA=1000;
 private static Object[] p;
 private static int muga=-1;

// Eraikitzailea (hasieraketa):
  private PilaEmaArray(){
    p=new Object[EDUKIERA];
  }

// Objektu kapsulatua
  private static PilaEmaArray pila=new
    PilaEmaArray();
```


EMA adibidea: pila

```
// Eragiketak

  public static int luzera(){
    return (muga+1);
  }

  public static boolean hutsaDago(){
    return (muga<0);
  }

  public static Object lehena(){
     return P[muga];
  }
```

EMA adibidea: pila

```
// Eragiketak:

  public static void pilaratu (Object o){
    P[++muga]=o;
  }

  public static Object despilatu(){
    Object elem;
    elem=P[muga];
    P[muga--]=null;
    return elem;
  }
}
```


Java programen antolaketa (IV): modifikatzaileen laburpena

	Ikusgaitasuna				***·Klaseen hedapena ·Eremu konstanteak ·Metodoen birdefinizioa***	***Kidearen partekotza*** *(heredatu ala ez)*	***Klase eta metodoen abstraktutasuna***
	package (besterik ezean)	`public`	`protected`	`private`	`final`	`static`	`abstract`
Klaseak	X	X	X	X	X		X
Eremuak (aldagaiak)	X	X	X	X	X	X	
Metodoak	X	X	X	X	X	X	X

Java programen antolaketa (V): modifikatzaileen laburpena

- Klaseen modifikatzaileak:
  - » `public`: edozein klasetatik atzigarria
  - » `private`: beste edozein klasetatik ez da atzigarria
  - » `package`: klasea definituta dagoen paketeko klaseetatik
  - » `protected`: `protected` gisa definitutako bere azpiklase guztietatik (pakete berekoak izan edo ez) eta pakete bereko klase guztietatik
  - » `static`: klase bat ezin da `static` izan!
  - » `abstract`: klase abstraktu bati dagozkion objektuak ezingo dira sortu, baina klase abstraktua hedatu ahal izango dugu
  - » `final`: klasea ezin da hedatu

Java programen antolaketa (VI): modifikatzaileen laburpena

- Eremuen modifikatzaileak:
  - » `public`: klasetik kanpora ere erabilgarria
  - » `private`: soilik erazagututa dagoen klasetik
  - » `package`: klasea definituta dagoen paketeko klaseetatik
  - » `protected`: definituta dagoen klasetik, azpiklase guztietatik (pakete berekoak izan edo ez) eta pakete bereko klase guztietatik
  - » `static`: klasearena da eremua: klase bateko objektu guztiek eremu bera konpartituko dute
  - » `abstract`: eremuak ezin dira abstraktuak izan!
  - » `final`: konstante bat izango da eremua

Java programen antolaketa (VII): modifikatzaileen laburpena

- Metodoen modifikatzaileak:
  - `public`: klasetik kanpora ere erabilgarria
  - `private`: soilik erazagututa dagoen klasetik
  - `package`: klasea definituta dagoen paketeko klaseetatik
  - `protected`: definituta dagoen klasetik, azpiklase guztietatik (pakete berekoak izan edo ez) eta pakete bereko klase guztietatik
  - `static`: klaseari dagokio, ez instantziei
  - `abstract`: ez dago inplementatuta klasean
  - `final`: ezin da gain-idatzi azpiklaseetan

Java programen antolaketa (VIII)

- Aplikazioak
  - `main` metodoa, aplikazioaren klase "nagusian"
- *Applet*-ak (web orrietarako)
  - Java klase-liburutegiko `Applet` klasea hedatzen da
  - `Applet` klaseak lau metodo garrantzitsu ditu:
    - `public void init () //applet-a kargatzen denean`
    - `public void start ()`
    - `public void stop ()`
    - `public void destroy ()`

Java programen antolaketa (IX)

- Paketeak:
  - Elkarrekin erlazionatutako klaseak biltzeko baliatzen diren “kutxa” modukoak dira.
  - Hierarkikoki antolatzen dira.
  - Klaseak definitzean esplizituki inportatzen dira paketeak.
- Interfazeak (`interface`):
  - Klaseen antz handia dute, baina metodoek ez dute gorputzik.
  - Klase batek interfaze bat edo gehiago inplementa ditzake: `implements` hedadura.

Java programen antolaketa (X)

- Javako iturburu-fitxategiaren egitura:
  - pakete-sententzia bakarra (aukeran)
    ```
    package pak1[.pak2[.pak3]];
    ```
  - inportazio-sententziak (aukeran)
    ```
    import pak1[.pak2].(klase-izena|*);
    ```
  - klaseen (publiko nahiz pribatuen) erazagupena

Java *vs.* Ada (I)

- Java-n ezin dira aldagai globalak erazagutu **:-o**
  - » Klaseko aldagai kideak klase osotik ikusten dira.
  - » Klase batetik kanpo ezin dira aldagaiak erazagutu.
- Java-n erregistrorik ez dago, baina klase bat bere atributuekin definitzen denean, hor dago erregistroaren ideia.
- **Informazioaren ezkutatzea:** Ada-n datu-mota eta eragiketen atzipena murriztu daiteke; Java-n eremu, klase eta metodoena.
- **Berrerabilgarritasuna:** Ez dago Java-n Ada-ko generikoen parekorik: ez da existitzen txantiloi edo "plantillaren" ideiarik, zeinetatik instantzia zehatzak sor daitezkeen parametro desberdinekin. Mota generikoak defini daitezke, nolabait, `Object` klaseaz baliatuz.

Java *vs.* Ada (II)

- **Modulartasuna:**
  - » Ada-ko paketeak Java-ko klaseen baliokide dira.
  - » Java-k ez du klaseen zehaztapena eta inplementazioa bereizten.
  - » Java-n paketeak erabiltzen dira klaseak multzokatzeko, eta Ada-n pakete-hierarkiak.
- **Herentzia**:
  - » Bietan dago herentzia, baina ez batean eta ez bestean askotariko herentzia. Java-n interfazeen bitartez simula daiteke. Ada-n zeharkako modu batzuk erabil daitezke antzeko gauzak lortzeko.
  - » Klase abstraktuen kontzeptua antzekoa da bietan.

Erabiltzaile-interfaze grafikoen diseinu-garapenak (GUI-ak)

- AWT paketea: elementuak.
  - » Edukiontziak.
  - » Antolaerak.
  - » Interfazeetako ageriko elementuak.
    - – Interakziozko osagaiak.
    - – Marrazten direnak.
    - – Animazioa.
    - – Irudiak.
- Gertaera-maneiatzailea: gertaerei zuzenduriko programazioa.

JDK (*Java Development Kit*)

- Interpretatzailea (*java*)
- Konpiladorea (*javac*)
- *Applet*-en ikuskatzailea (*appletviewer*)
- Araztailea (*jdb*)
- Dokumentazioen sortzailea (*javadoc*)

Amarauneko helbide interesgarriak

- http://java.sun.com (ingelesez)
- http://javaboutique.internet.com/ (ingelesez)
- http://softwaredev.earthweb.com/java (ingelesez)
- http://www.programacion.com/java/ (gaztelaniaz)
- http://www.javahispano.com/ (gaztelaniaz)
- http://www.monografias.com/trabajos/java/java.shtml (gaztelaniaz)

Bibliografia

- *El lenguaje de programación Java.* Arnold, K. *et al.* Addison Wesley, 2001.
- *Java 2. Manual de usuario y tutorial.* A. Froufe. Ra-Ma, 2000.
- *Descubre Java 1.2.* M. Morgan. Prentice-Hall, 1998.
- *Problemas resueltos de programación en lenguaje Java.* Pérez Menor, J.M. *et al.* Thomson, 2003.
- *An introduction to Data Structures and Algorithms with Java.* G.W. Rowe. Prentice-Hall, 1998.
- *Estructuras de datos en Java.* Weiss, M.A. Addison Wesley, 2001.