JAVA

1. Salbuespenak
2. Fitxategiak gorde
3. Array-ak
4. ~~ArrayList~~
5. Objektuetara bideratuta programazioa
6. Java to XML

1. Salbuespenak

- Programa bat exekutatzen denean ematen dituen erroreak dira Salbuespenak

- Javan, aurretiaz zehaztu genezake zer egin salbuespenak sor ditzaketen kode-zatiak exekutatzean:
    - errorea gertatuko balitz, programaren exekuzio fluxua aldatu egingo litzateke, eta salbuespen horri erantzuna emango dion kode-bloke batetara helduko litzateke.
    - Salbuespena harrapatzen ez badugu, programa gelditu egingo da kasurik gehienetan.

1. Salbuespenak: Hierarkia diagrama

1 Salbuespenak: Nola arrapatu

- Salbuespenak arrapatzeko **try-catch-finally** erabiliko dugu

- Salbuespena harrapatu ahal izateko, salbuespen hori eragin dezaketen sententziak try bloke baten barruan sartu behar dira.

- Salbuespena gertatzen bada, salbuespen hori jaurti egingo da, eta catch blokeek harrapatu egin dezakete jaurtitako salbuespen hori.

- Bloke horien barruan kudeatuko dira salbuespenak.

1. Salbuespenak: Sintaxia

Erabiltzen den sintaxia hau da:

```
try {
 salbuespena sortu dezakeen kode-zatia
} catch (Salbuespen_mota_1 salbuespen_objektua) {
 Salbuespen_mota_1 kudeatzea;
} catch (Salbuespen_mota_2 salbuespen_objektua) {
 Salbuespen_mota_2 kudeatzea;
} ...
finally {
 Beti exekutatuko den kodea.
}
```

catch zatia hainbat aldiz errepika daiteke; kudeatu nahi dugun salbuespen mota bakoitzeko *catch* bat jarriko dugu.
Finally zatia hautazkoa da, eta, agertzekotan, behin bakarrik agertuko da. *catch* bakoitzak salbuespen mota bat kudeatzen du.

Salbuespena gertatzen denean, salbuespenkudeatzaile egokia daukan *catch*-a bilatuko du programak, hau da, gertatu den salbuespenaren mota berekoa dena.
Kontu handiz ibili behar dugu, garrantzi handikoa baita *catch*-ak zer ordenatan dauden idatzita. Adibidez, demagun ***ArithmeticException*** motako salbuespen bat gertatzen dela, eta lehenengo postuan kokatuta daukagun *catch*-ak ***Exception*** motako salbuespenak harrapatzen dituela. *ArithmeticException* salbuespena *Exception superklasearen azpiklase* bat da; beraz, lehenengo *catch* hori besterik ez da exekutatuko.

2. Fitxategiak gorde

- Sarrera fluxua kudeatzen duten klaseak “Writer” klase abstraktutik heredatzen dute.
- IOException salbuespenak egon daitezke
- Write klasearen oinarrizko metodoak ondorengoak dira:

void write (int c)	Karaktere bat idatzi
void write (String cadena)	Kate bat idatzi
void write (char [] array)	Karaktere array bat idatzi
void flush ()	Writer-eko irteera fluxua uztu
void close ()	Writer-eko irteera fluxua uztu eta itxi

2. Fitxategiak gorde

- Writer clase abstraktua erabiltzen duten klase ezagunetarikoenak ondorengoak dira:
  - FileWriter
  - BufferedWriter
  - PrintWriter
  - StringWriter
- PrintWriter → erabiliko dugun klasea ondorengo ezaugarriak ditu
  - Edozein irteera formatuan idatzi ditzake karaktereak
  - System.out-en bezala print, printf eta println erablitzen ditu idazteko fitxategiak
  - flush() edo close() egin arte ez du ezer idatziko

2. Fitxategiak gorde: PrintWriter adibidea

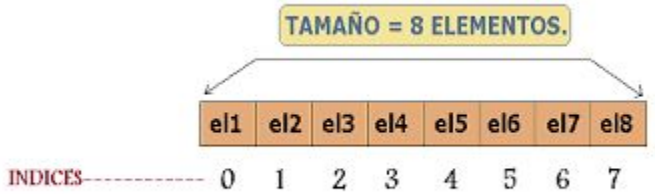
```
FileWriter f = new FileWriter (“bidea fitxategira”); // new FileWriter (“bidea fitxategira”, true / false); //aukera ematen du editatzeko

PrintWriter pw= null; // Salbuespenetik kanpo sortu, erabilgarri egoteko

try {

        PrintWriter pw = new PrintWriter (f); // sortzen dugu fitxategiarekin

        // idazteko aukera desberdinak

        pw.println (“Fitxategian idazten”);

        pw.printf (“Idaztern %d lerroa formatuarekin eta lerro saltoa \n”,2);

        pw.flush();

        pw.close();

        f.close();

} catch  (IOException ex) {

        Salbuespena kudeatzeko lerroak

}
```

3. Array-ak

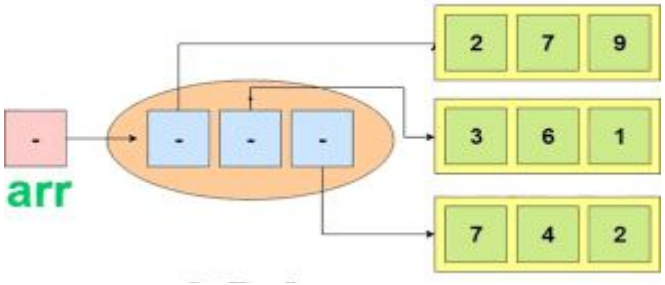
- Orain arte, sortutako aldagai bakoitzeko datu bakar bat gorde izan dugu
  - Baina zer gertatzen da programa batek datu kopuru handia erabili behar duenean? dakigunarekin, datu bakoitzeko aldagai bat sortu beharko genuke
  - Javan, beste programazio lengoaietan bezala, beste elementuen biltegiratze bezala egiten duten aldagaiak dituzte. Hauek dira Array-ak
- Array bat izan daiteke estatikoa, bere elementu kopurua finkoa da sortzen den momentutik
- Edo dinamikoa, elementu kopurua aldakorra duena.

3. Array-ak: Dimentsioak

- Array batek dimentsio desberdinetan gorde ditzake datuak:
  - Dimentsio bakarreko array-a:



  - Array multidimentsionala: array baten elementu bakoitza array bat da.



3. Array-ak: Adierazpena eta sorrera

- Array bat adierazita izango da esanez elementuaren datu mota, dimentsioa eta aldagaiaren izena.
- Kortxete bikoite bakoitzak ([]) adieraziko du dimentsio bat. Posible da izena eta kortxeten ordena aldatzea
- Adib:
  - short [] adinak;
  - int [] pisuak;
  - int salneurriak []:
  - int [] [] taula;
  - byte [] [] [] hiruDimentsiotakoArraya;
- Behin adierazita memorian espazioa gorde behar da gordetu dituen balioentzako:
  - adinak = new short [10];
  - pisuak = new int [50];
  - salneurriak = new int [100];
  - taula = new int [4][2];
  - hiruDimentsiotakoArraya = new byte [4][3][10];

3. Array-ak: Abiarazi eta Sarbidea

- Array bat abiarazteko datu bat esleitu behar zaio.
- Array baten elementu baterako sarbidea indize baten bidez da.
- Array-aren tamainatik kanpoko indize bat jartzekotan “java.lang.ArrayIndexOutOfBoundsException” salbuespena agertuko zen.
- Adib:
  - adinak[0]=1;
  - pisua [1]=3;
  - salneurriak [1]=2;
  - taula [2][1] = 200;
  - hiruDimentsiotakoArraya [2][2][0] = 127;

3. Array-ak: Metodoak

- Array bat bidalia izan daiteke argumentu bezala metodo batean.

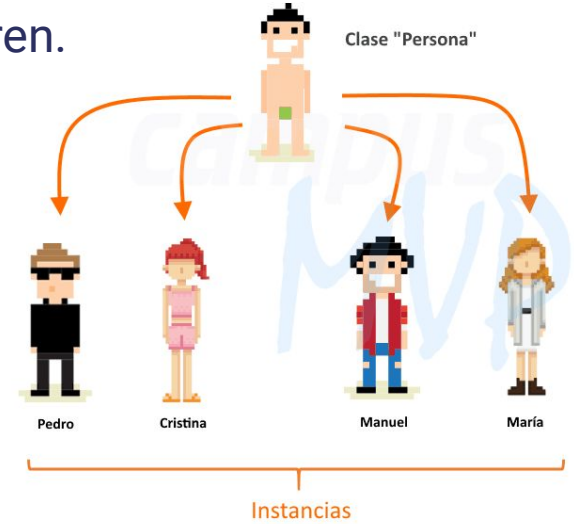
  int [] [] itzuleraBalioak = adibidea (adinak, pisuak, salneurriak, hiruDimentsiotakoArraya);

- Edo metodo baten itzulera-balioa bezala:

```
public static int [] [] adibidea ( short [] array1, int [] array2, int [] array3, byte [][][] array4) {
        // …
        int [] [] itzuleraArraya = new int [4][2];
        // …
        return itzuleraArraya;
}
```

OBP: Objektuetara bideratutako programazioa

- Programazio egituratua bezala, objektuetara bideratutako programazioa (POO) aginduzko programazioaren paradigmatik jaisten den programazio mota bat da.
- Objektuetara bideratutako programazioa mundu errealaren abstrakzio batean oinarritzen da, non objektuak taldeetan sailkatzen diren.



OBP: Objektuen propietateak eta metodoak

Objektu batek parte hauek ditu:

- **Atributuak** (Propietateak): Datuak gordetzen dituen objektuaren atala. Aldagai klasekidea ere esaten zaie. Datu hauek oinarrizko datu motetakoak izan daitezke (boolean, char, int, double...) edo beste objektu mota batekoak. Adibidez, Auto motako objektu batek Gurpil motako objektuak izan ditzake.
- **Metodoak** (Funtzio klasekideak): Atributuekin eragiketak egiten dituen objektuaren atala.

OBP: Klaseak

- Klasea da egitura eta portaera berdina daukaten hainbat objekturen deskripzioa.
- Klasetik abiatuta sortzen dira beharrezkoak ditugun kopiak edo instantziak. Kopia horiek dira klasearen objektuak.
- Klaseek datuak eta metodoak dauzkate. Datu eta metodo horiek objektu multzo batek dituen ezaugarriak definitzen dituzte.
- Programa bat klasez osatuta dago, eta klase horietatik abiatuta eratzen dira objektuak. Objektu horiek, gero, interakzionatu egingo dute elkarren artean.
- Beste era batera esateko, klase bat eredu bat da.

OBP: Klaseak adibidea

main() metodoa, programan, behin bakarrik jartzen da, klase nagusian. Metodo horretatik hasten da programa exekutatzen.
Sortzen ari garen klasea ez bada klase nagusia, metodo hori ez da agertuko.

```
//Klasearen burua
public class Pertsona {
        //Klasearen gorputza
        String izena;
         String abizenak;
        int adina;
        double garaiera;
        double pisua;
        public Pertsona() {}
        public static void main(String[] args) {
        // TODO Auto-generated method stub
        }
}
```

OBP: Objektuak deklaratzea eta sortzea

- Objektu bat sortzeko, urrats hauek eman behar dira:
  - Deklaratzea: Objektu mota definitzea.
  - Instantziatzea: Objektua sortzea (new eragilea erabilita).
- Baina nola egiten dira pauso horiek?
  - Objektu bat deklaratzeko, hau egin behar da:
    - <klasea> objektuaren izena
  - Baina hori egitea ez da nahikoa objektua sortzeko. Objektuaren erreferentzia sortu besterik ez dugu egin; baina erreferentziaren balioa hutsik dago (null balioa dauka).
  - Behin erreferentzia sortuta daukagunean, erreferentzia horretan gordeko den instantzia edo objektua sortu behar dugu. Horretarako, new eragilea erabiliko dugu:
    - objektuaren izena = new <klaseko sortzailea>(parametroak)
- Adib:
  - Pertsona p; // deklaratu
  - p = new Pertsona (parametroak); // sortu

OBP: Atributuak

klasea definitzen duten datuei atributuak ukatzen zaizkie

Adibidez: Ibilgailu klasea definitu daiteke matrikula, kolorea, marka eta modeloarekin

Nola definitu atributuak:

```
class klasearenIzena {
    mota atributu1;
    mota atributu2;
    final mota atributu3; // behin balioa emanda ezin izango da aldatu
    …
}
```

OBP: Metodoak

Metodoek klasearen portaera definitzen dute.

Klase bateko objektuek egin ditzaketen portaerei edo opreazioei metodoak esaten zaie.

Adibidez:

```
public class klasearenIzena {
    // atrubutuak adierazi
    mota metodoarenIzena (parametroak) {
        // kodea
    }
}
```

Persona
izena adina altuera
agurtu() urteakBete() hazi(zenbat)

OBP: Eraikitzaileak

Zein balio hartzen ditu sortu berri den objektu bat? → sortzerakoan baliorik esleitzen ez zaien atruibutuak lehenetsita hasten dira, motaren arabera, honela:

- 0 zenbakidun balioei eta char
- null erreferentziak
- false bolearrak

Hala ere, orokorrean, objektu bat erabiltzen hasi baino lehenago nahi izango dugu atributuei balioak ematea. Horretarako eraikitzaileak erabiliko ditugu

Adibidea: Pertsona klasea

OBP: Enkapsulatzea (getter eta setter)

- Klaseko atributuak eta metodoak hobeto kontrolatzea.
- Klase-atributuak irakurgai egin daitezke, bakarrik (get metodoa bakarrik erabiltzen bada), edo idatzi (set metodoa bakarrik erabiltzen bada).
- Malgua: programatzaileak kodearen zati bat alda dezake beste zati batzuei eragin gabe.
- Datuen segurtasun handiagoa

get metodoak atributuaren balioa itzultzen du eta set metodoak balioa ezartzen du atributuan.

Bata eta bestearen sintaxia honako hau da:

- get edo set atributuaren izenari jarraituz lehenengo letra handiz

```
public class Pertsona{
  String izena;

  // Getter
  String getIzena() {
    return name;
  }

  // Setter
  void setIzena(String i) {
    izena= i;
  }
}
```

OBP: Sarbide-aldatzaileak (Metodoak, Atributuak eta Sortzaileak)

Public	Kodea eskuragarri dago klase guztientzat.
Private	Kodea adierazitako klasearen barruan bakarrik erabili daiteke.
Protected	Kodea bakarrik subklasean erabili daiteke

```
public class Pertsona{
  private String izena;

  // Getter
  public String getIzena() {
    return name;
  }

  // Setter
  public void setIzena(String i) {
    izena= i;
  }
}
```

OBP: THIS Hitz gakoa

this gakoak metodo edo eraikitzaile baten objektuari egiten dio erreferentzia.

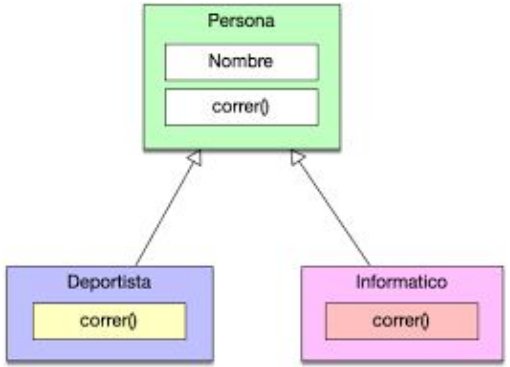
Thin hitz giltzarri honen erabilerarik ohikoena klaseko atributuen eta izen bereko parametroen arteko nahasketa ezabatzea da (klaseko atributu bat metodo edo parametro eraikitzaile batek zapaltzen duelako).

Aurreko adibidean gako-hitza alde batera utziz gero, irteera "0" izango litzateke "5" ren ordez.

```
public class Main {
  int x;

  // Sortzailea parametroarekin
  public Main(int x) {
    this.x = x;
  }

  // Deitu sortzailea
  public static void main(String[] args) {
    Main obj = new Main(5);
    System.out.println("Value of x = " + obj.x);
  }
}
```

OBP: Herentzia

- Ezaugarriak gurasoetatik seme-alabetara igarotzea ahalbidetzen du
- Klase batek beste baten oinordeko denean, bere ezaugarri eta metodo ikusgarriak bereganatzen ditu, eta kodea eta funtzionalitateak berrerabiltzeko aukera ematen du.
- Heredatzen den klaseari guraso-klasea edo superklasea esaten zaio, eta heredatzen duen klaseari alaba-klasea edo azpiklasea.



OBP: Subklasea eta superklasea

- Azpiklase batek superklasetik heredatutako kideak ditu eta, atributu eta motodo berriak gehitzen zaizkio.
- Horrek funtzionaltasuna handitzen du, eta, aldi berean, kodea alferrik errepikatzea saihesten du.
- Java klase guztiek Object klasetik heredatzen dute, APIn ere definitua.
- Zelan egin superklasetik heredatzeko?

```
class SubKlasea extends SuperKlasea {



}
```

OBP: Herentzia bidezko sarbide-aldatzaileak

	Ikusgarria nondik …			
	Klasetik	Auzo-klaseak	Azpi-klaseak	Kanpoko klaseak
private	✔			
aldatzailerik ez	✔	✔		
protected	✔	✔	✔	
public	✔	✔	✔	✔

XML: Java to XML

- XML motako testu-fitxategien bidez, gero eta ohikoagoa da informazioa gorde eta transmititzea.
- Informazio hori programazio-lengoaiekin prozesatzeko, Javak JAXB (Java Architecture for XML Binding) APIa inplementatu du
- XML fitxategi baten elementuak Javaren klase eta objektu multzo batekin lotzeko aukera ematen du, bi zentzuetan bihurketak eginez.
- Java 8. bertsioan, berriagoak direnekin “deprecated”

- Elementuak Javan lan egiteko XML fitxategiekin:
  - **import javax.xml.bind.annotation:*;** // xml anotazioen paketea
  - Klasea definitu baino lehenago
    - **@XmlRootElement (name=”xml erro dokumentuaren izena”)**
    - **@XmlType (propOrder = {“klaseko aldagaien izenak”, “aldagaia”}) //klaserako aldagaiak definitu**
    - **@XmlAccessorType (XmlAccessType.FIELD) //sarbide mota**
  - Atributu bakoitzaren gainean:
    - **@XmlAttribute (name=”izena”, required=true)**
    - **@XmlElement (name=”izena”)**

XML: Klasea definitu Adibidea

XML adibide bat:

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”yes”?>

<socio id=”123”>

    <nombre>Martin Fisher</nombre>

    <direccion>43 Bass St</direccion>

    <alta>12/12/2022</alta>

</socio>
```

Sortu klasea xml-aren esturkturan oinarrituta.

```
import javax.xml.bind.annotation.*
//sortu klasea
@XmlRootElement (name=”socio”) //erro elementua
@XmlType (propOrder = {“nombreSocio”, “direccion”, “fechaAlta”})
@XmlAccessorType (XmlAccessType.FIELD)
public class Socio {
        @XmlAttribute (required=true)
        private int id;
        @XmlElement (name=”nombre”)
        private String nombreSocio;
        @XmlElement (name=”direccion”) // ez da beharrezkoa izen berdina duelako xml dokumentuan
        private String direccion;
        @XmlElement (name=”alta”)
        private String fechaAlta;

        public Socio(){ }
        public Socio (int id, String nombreSocio, String direccion, String fechaAlta){
                this.id=id;
                this.nombreSocio=nombreSocio;
                this.direccion=direccion;
                this.fechaAlta=fechaAlta;
        }
        // falta dira getter eta setter-ak eta toString

}
```

XML: Unmarshall. Klase nagusia

- import javax.xml.bind.JAXBContext;
- import javax.xml.bind.JAXBException;
- import javax.xml.bind.Unmarshaller;
- import javax.xml.bind.PropertyException;
- import java.io.File;

- JAXBContext c = JAXBContext.newInstance (“Klasea”.class);
- Unmarshaller um = c.createUnmarshaller();
- Irakurri fitxategia (salbuespena kontrolatu behar da)
  - Kasting-a egin behar da bueltatzen duela Object datu mota.
  - “klasea” k = (Klasea) um.unmarshal (new File(“fitxategia.xml”));

XML: Marshall. Klase nagusia

- import javax.xml.bind.JAXBContext;
- import javax.xml.bind.JAXBException;
- import javax.xml.bind.Marshaller;
- import javax.xml.bind.PropertyException;
- import java.io.FileWriter;

- Marshaller m = c.createMarshaller();
- Irteera formatuarekin emateko setProperty ezarri behar da, bestela dena lerro batean egongo da.
- m.setProperty (Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
- m.marshal (“aldagaia”, new FileWriter (“izena.xml”);

XML: Elementu askorekin lan egin (Wrapper)

- Zer egin XML batean elementu asko ditugunean?
  - wrapper-en bitartez datuak bilduko dira

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”yes”?>
<socios>
    <socio id=”1”>
        <nombre>Martin Fisher</nombre>
        <direccion>43 Bass St</direccion>
        <alta>12/12/2022</alta>
    </socio>
    <socio id=”2”>
        <nombre>John Lewis</nombre>
        <direccion>54 Down St</direccion>
        <alta>12/12/2022</alta>
    </socio>
</socios>
```

```
@XmlElementWrapper (name = “socios”)
@XmlElement (name = “socio”)
private Socio [] listaSocios;
```