

# Hlutbundin forritun með Java

Ólafur Andri Ragnarsson  
HR

# Efnistöð

- Stuttur inngangur um Java
- Tilvísunartög
- Hlutbundin forritun
- Klasar
- Erfðir
- Skil
- Undantekningar

Stuttur inngangur um Java

# Java forritunarmálið

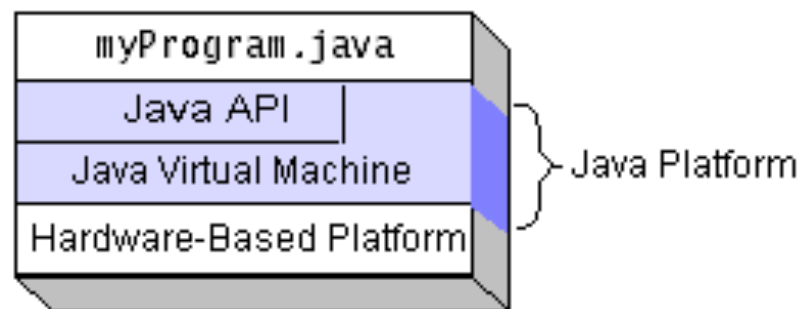
- Java er einfalt, fágæð, og auðlært mál
  - Hlutbundið forritunarmál
  - Svipar til C# sem byggir mikið á Java
  - Java er C++ án flækjustigsins
- Mikilvægir þættir innbyggðir
  - Minnisstjórnun (ruslasöfnun)
  - Þráðavinnsla
  - Netvinnsla
  - Skil
- Þó hafa ýmsir gagnrýnt málið
  - Breytingar á málinu hafa komið fram

# Java sýndarvélin

- Java forrit keyra á sýndarvél
  - Java Virtual Machine (VM)
  - Ein sýndarvél fyrir hvert stýrikerfi
  - Gamall draumur úr Unix heiminum – alheimsvélin
  - Bera saman við Win32 og Windows CLR
- Kostir
  - Forritin keyra “allsstaðar” – WORA
- Umræðan
  - Sumir telja JVM það mikilvægasta við Java

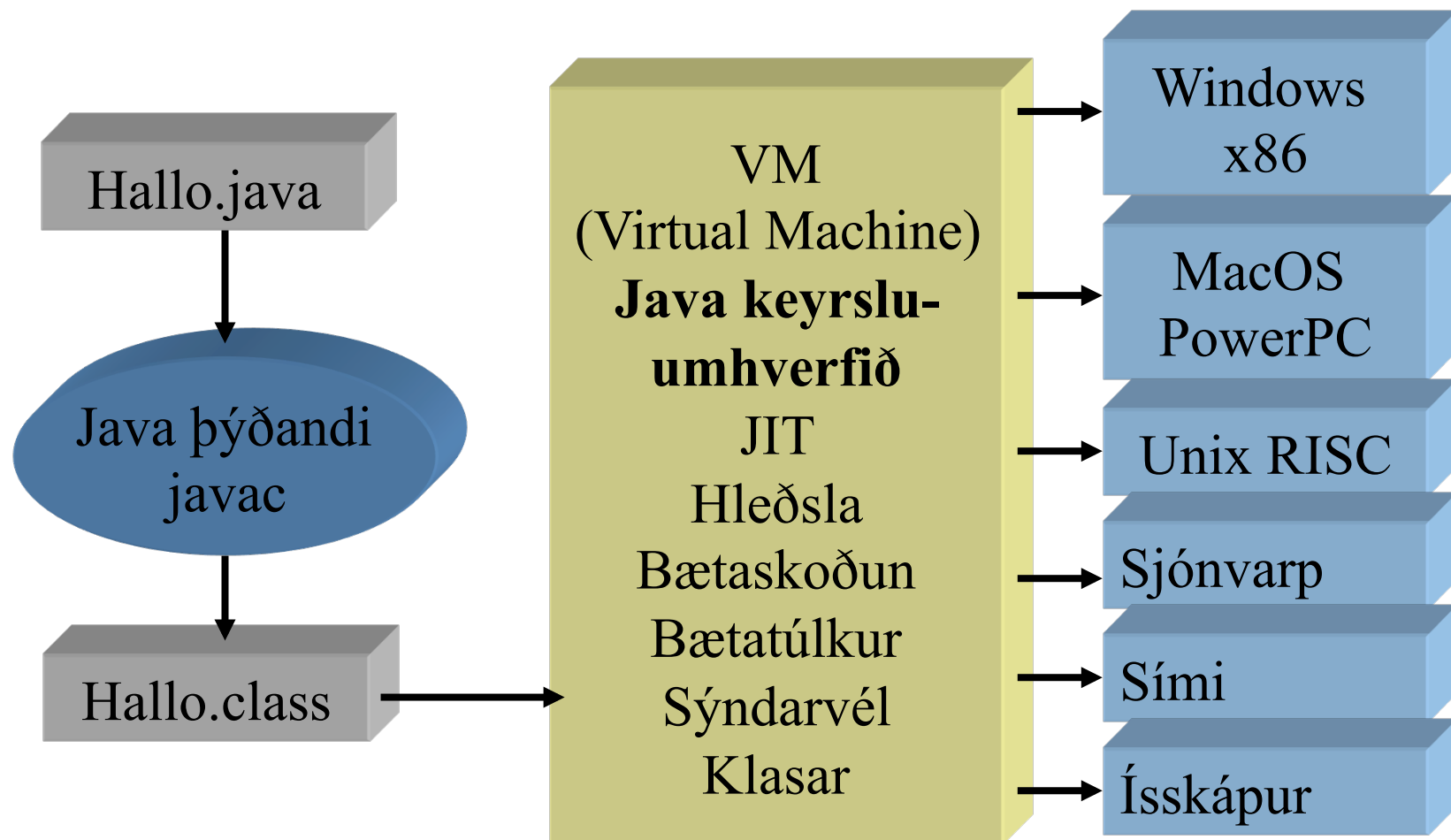
# Java grunnkerfið – platform API

- Grunnkerfið er samansafn af APlum
  - Platform API (dæmi: util, lang, text, io, net)
  - Þróun Java undanfarin ár liggur í grunnkerfinu
  - Bera saman við Win32 API, MFC og .NET
- Klasasöfn gera forritun einfaldari
  - Felur undirliggjandi stýrikerfi

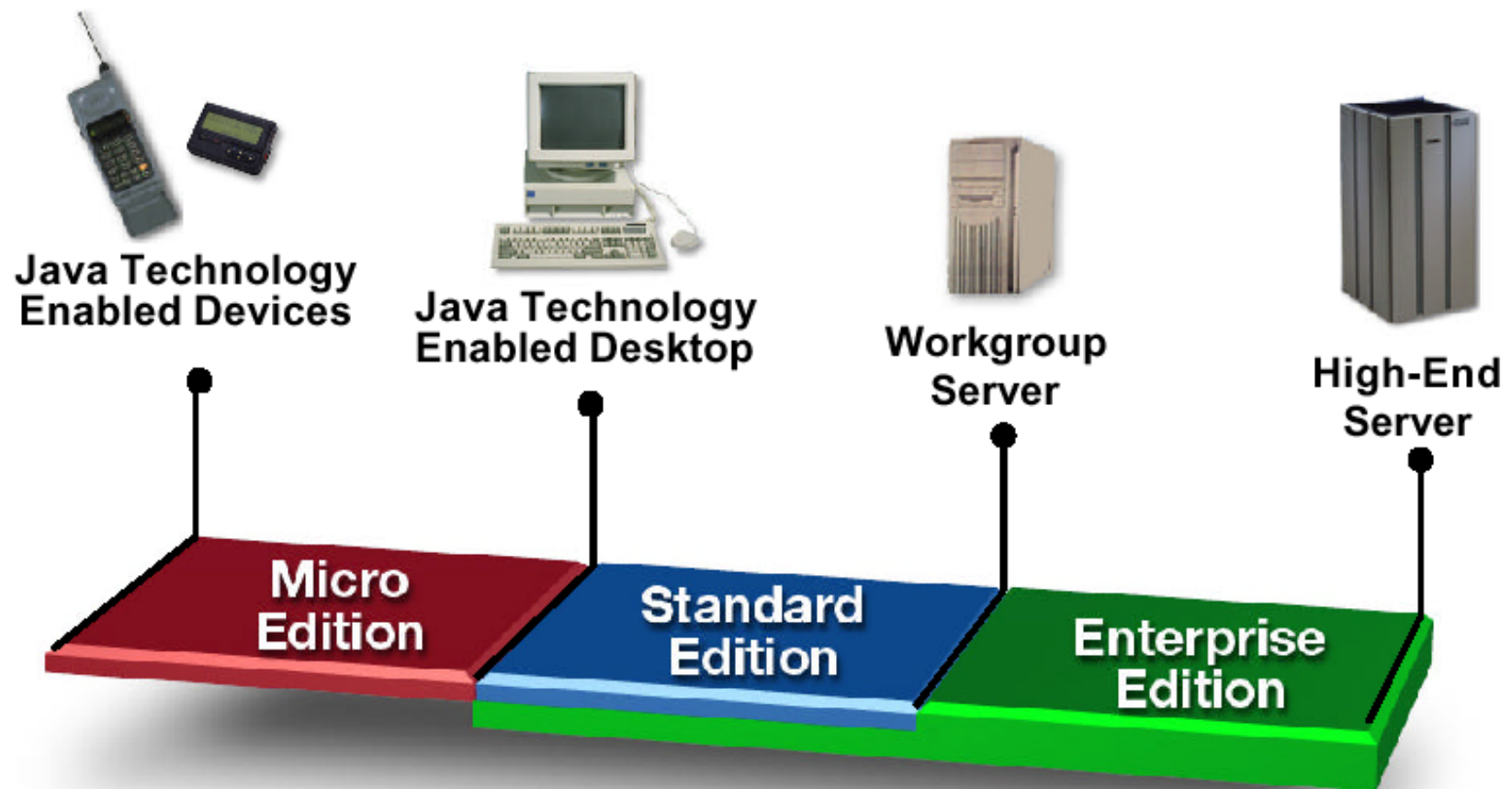


# WORA hugmyndin

- Write-Once-Run-Anywhere



# Java Platform





# Java útgáfan af HelloWorld

- Hver public klasi er í sér Java skrá
  - Skráin heitir það sama og klasinn, t.d. *HelloWorld.java*

```
import java.lang.*;
```

```
public class HelloWorld
{
    public void sayHello (String s)
    {
        System.out.println ("Hello " + s);
    }

    public static void main (String args[])
    {
        HelloWorld hello = new HelloWorld ();
        hello.sayHello("World");
    }
}
```

Tilvísunartög

# Tilvísunartög (reference type)

- Tög sem ekki eru grunntög eru tilvísunartög
  - Vísa á tilvika af klasa eða array
- Tilvísunartög eru bendar á tilvik

```
Point p,q;  
p = new Point ();  
q = p;
```

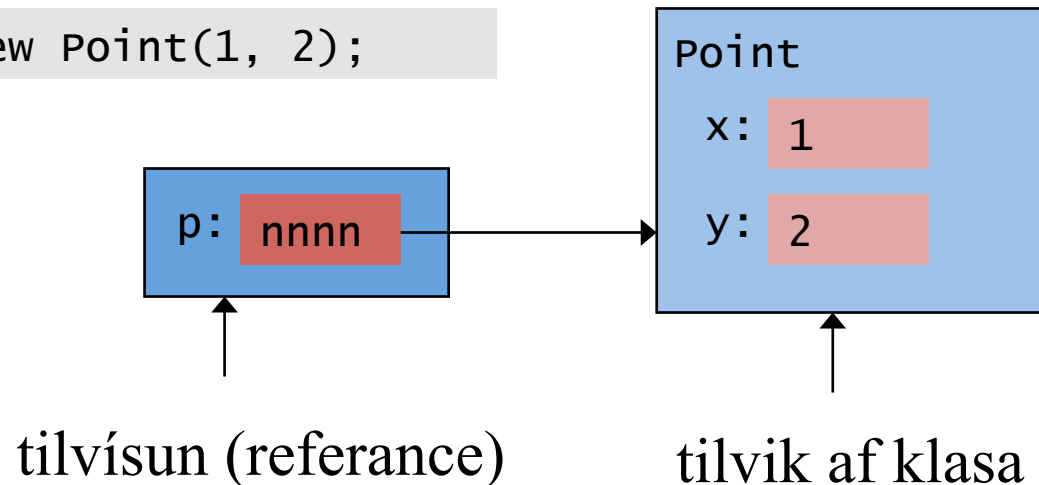
- Færíbreytur í föllum
  - Grunntög: “call-by-value”
  - Array og tilvísunartög: afrita tilvísunina ekki tilvikið sjálft

# Tilvísunartög

- Tilvísunartög vísa á tilvik af klösum eða array
  - Staerð tilviks er óskilgreint

*new* býr til minnissvæði fyrir klasann

```
Point p = new Point(1, 2);
```

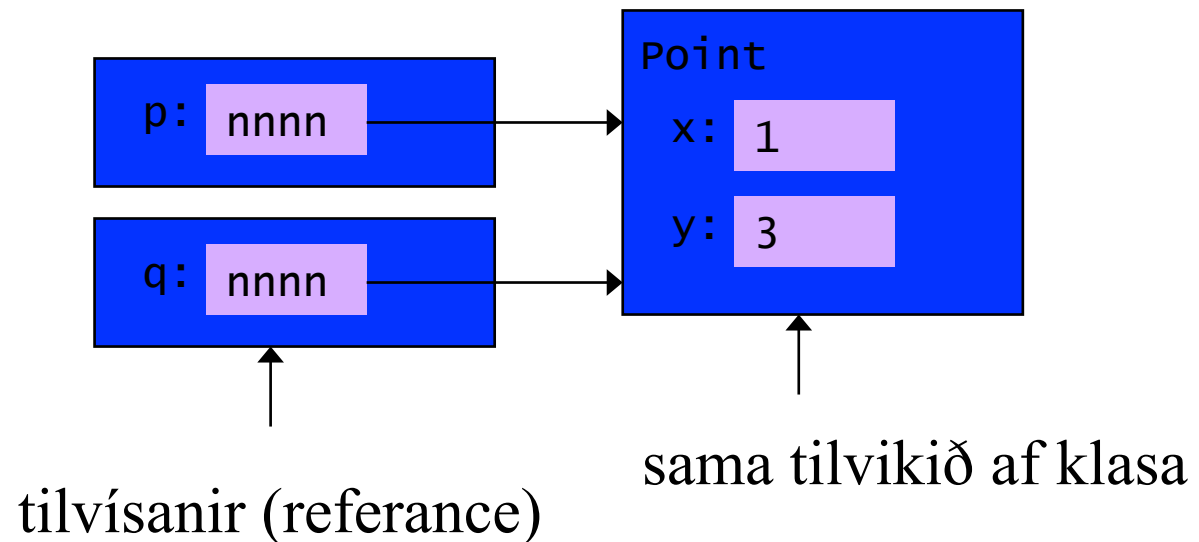


*p* er tilvísunartag, tilvikið er af taginu *Point*

# Tilvísunartög

- Afritun býr til nýja tilvísun, ekki nýtt tilvik

```
Point p = new Point(1, 3);  
Point q = p;
```

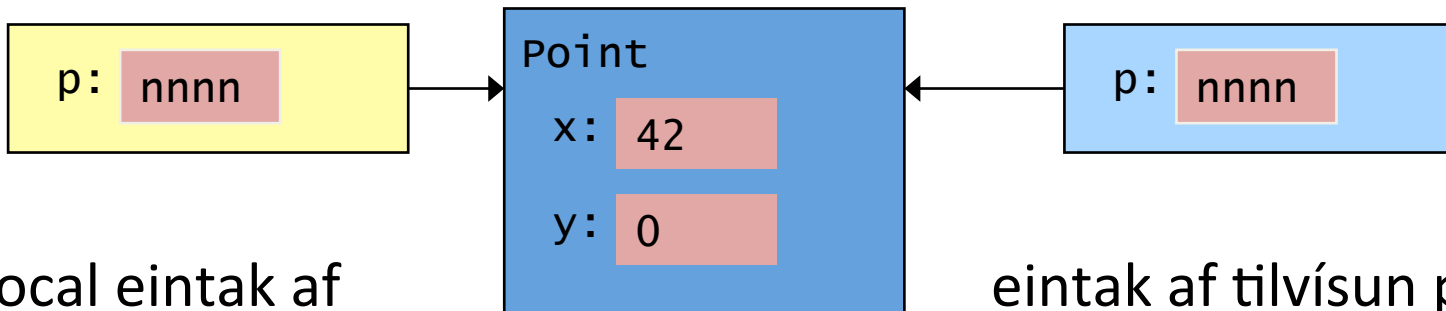


# Tilvísunartög

- Færíbreytur eru afrit af tilvísunum sem vísa á sama tilvikið

```
void changeReferance(Point p)
{
    while (p.x > 0)
        p.x--;
}
```

```
Point p = new Point (42,0);
changeReferance(p);
System.out.println(p.x);  // 0
```



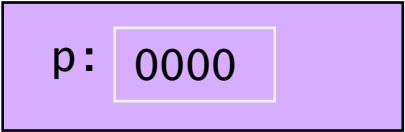
local eintak af  
tilvísun *p* innan  
*changeReferance*

eintak af tilvísun *p* sem  
bendir á Point tilvik

# NullPointerException

- Algeng undantekning í nemendaverkefnum
- Ástæðan
  - *new* virkinn býr til minnissvæði og tilvik
  - Ekki er hægt að nota breytu sem er af tilvísunartagi nema að búið sé að *new*'a klasa til að fá raunverulegt tilvik af klasanum
- Dæmi:

```
Vector v = null;  
v.add ("First");
```



A purple rectangular box representing a memory segment. Inside, on the left, is the text 'p:'. To its right is a smaller white rectangular box containing the text '0000'.

```
C:\Java2>javac npex.java
```

```
C:\Java2>java npex
```

```
Exception in thread "main" java.lang.NullPointerException  
at npex.main(npex.java:10)
```

Hlutbundin forritun



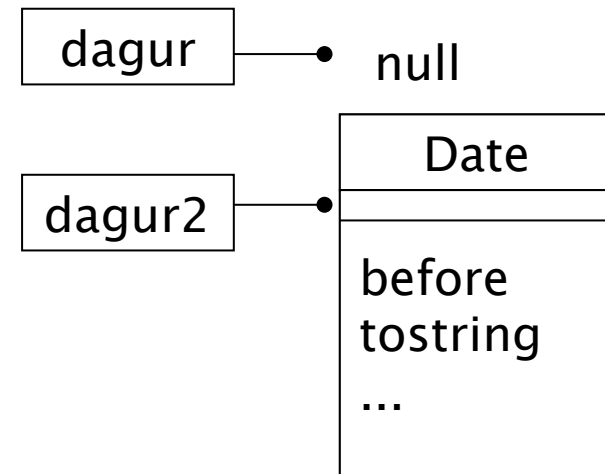
# Hlutbundin forritun

- Java er hlutbundið forritunarmál
  - Tilvik (object) eru búin til með *new*
  - Tilvísunarbreytur (object variables) benda á tilvik
- Hlutbundin fræði
  - Klasar eru lýsingar á hvernig tilvik eiga að vera
  - Klasar erfa aðra klasa (inheritance)
  - Klasabreytur (instance variables) eru huldar (encapsulated)
  - Klasar hafa föll (methods) sem veita aðgang í breytur klasans

# Hlutbundin forritun

- Tilvik búið til

```
Date dagur;  
Date dagur2 = new Date ();  
  
System.out.println (new Date ());  
String s = new Date().toString ();
```



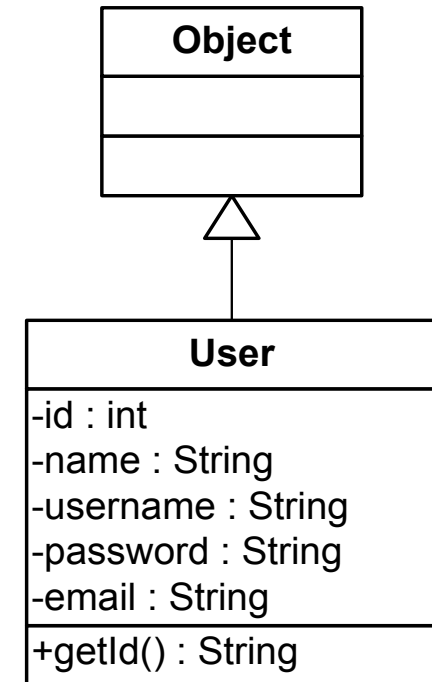
- Tilvik notað

```
nuna = new Date();  
if (dagur2.before(nuna))  
{  
    System.out.println (dagur2.toString());  
}
```

Klasar

# Klasar

- Allir klasar erfa frá öðrum klösum
  - Aðeins einfaldar erfðir
  - Klasinn *Object* er grunnklasinn
    - Sjálfgefið ef ekki tekið fram
  - *Final* klasa er ekki hægt að erfa
- Klasi hefur
  - Klasabreytur (instance variables)
  - Klasaföll (methods)
- Grunnklasinn *Object* bendir á hvaða tilvik sem

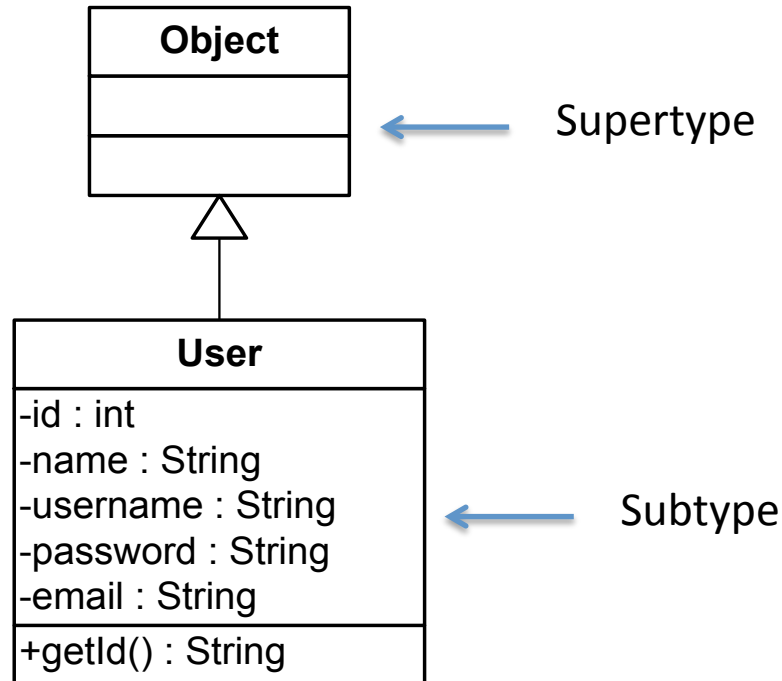


```
Object obj = new Date ();
Date d = (Date)obj;
```

```
Class User extends Object
```

# Klasar

- **Object** er super-type fyrir **User**
- **User** er sub-type af **Object**



# Klasaföll

- Klasaföll má *yfirskrifa* (override)
  - Klasi erfir klasa og yfirskrifar fall
- Klasaföll má *yfirhlaða* (overload)
  - Sama fallið tekur mismunandi færibreytur
- Klasaföll geta verið
  - *public* – allir geta keyrt
  - *private* – aðeins keyrt í sama klasa
  - *protected* – keyrt í sama klasa og erfðum klösum

## Klasaföll – smiður

- Klasi hefur smið (constructor)
  - Ber sama nafn og klasinn
  - Getur tekið hvaða færribreytur sem er, líka enga
  - Skilar engu
- Alltaf kallað á smið superklasa
  - “constructor chaining”
  - Þýðandinn setur inn sjálfgefinn smið ef hann er ekki skilgreindur
  - Ef skilgreindur er smiður annar en sjálfgefinn, er ekki hægt að keyra sjálfgefin smið

# Klasatilvísanir – *this* og *super*

- *this* – táknar klasann sjálfan

```
public Employee (String name, double salary)
{
    this (name);    // verður að vera fyrst
    this.salary = salary;
}
```

- *super* – táknar klasann sem erft er frá

```
public Manager (String name) // erfir Employee
{
    super (name, 0.0);    // verður að vera fyrst
    ...
}
```



# Dæmi

```
class Point
{
    private int x, y;
    public Point ()
    {
        x = 0; y = 0;
    }
    public Point (int x, int y)
    {
        this.x = x; this.y = y;
    }
    public void move (int dx, int dy)
    {
        x+=dx;y+=dy;
    }
    public String toString ()
    {
        return "(" + x + "," + y + ")";
    }
}
```

← **Klasabreytur**

← **Sjálfgefinn smiður (default)**

← **Yfirhlaðinn smiður**

← ***this* notað til að aðgreina milli færíbreytu og klasabreytu**

← **Yfirskrifum *toString* fallið í *Object*. Á að skila upplýsingum um tilvikið**

# Dæmi

```
public class Test
{
    public static void main (String[] args)
    {
        System.out.println ("Test");
        Test test = new Test();
    }
    public Test ()
    {
        Point p0;    // null referance
        Point p1 = new Point ();
        Point p2 = new Point (1,2);

        Object obj = p2;
        p0 = (Point)obj;
        p0.move (1, 1);
        System.out.println("p0=" + p0);
        p0 = null; // ekki notað lengur
    }
}
```

```
C:\java>javac Test.java
```

```
C:\java>java Test
```

```
Test
```

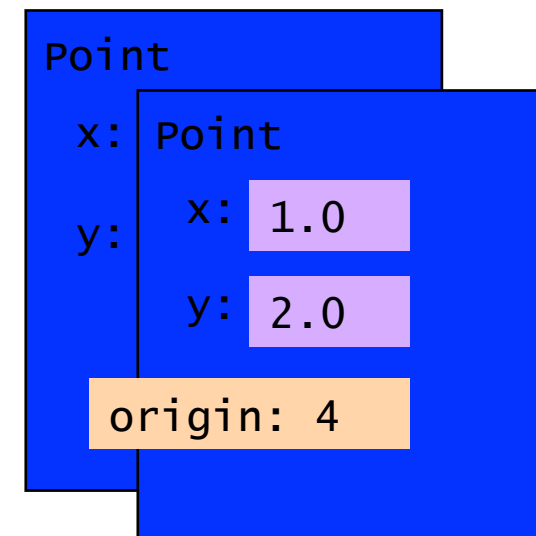
```
p0= (2,3)
```

# Static breytur og föll

- Breytur og föll geta verið “static”
  - Eiga þá við klasann en ekki tilvik hans
  - Static breyta er sú sama í öllum klösum
  - Static föll vinna ekki á tilvikum og eru aðgengileg án tilviks af klasa

```
class Point
{
    static int origin_;
    static void setOrigin(int origin)
    {
        origin_ = origin;
    }
}

Point.setOrigin(4);
int n = Math.abs (-3049);
```



# Static föll - main

- main fallið

- *public static void main (String args[])*

```
public class Employee
{
    ...

    public static void main (String args[])
    {
        Employee e = new Employee ("Dilbert", 100000, new Date());
    }
}
```

- Allir klasar geta verið með main fall
  - Hentugt fyrir einingaprófanir

# Synchronized föll

- Merkjum föll *synchronized* þegar læsa þarf aðgangi í fall
- Læsing til að tryggja aðgang í breytur
- Aðeins eitt tilvik getur keyrt fall í einu

```
class Database
{
    synchronized void writeData ()
    {
        ...
    }
}
```

# Singleton Registry (480)

- Aðeins eitt tilvik til af klasa

```
public class Registry
{
    private static Registry soleInstance = new Registry();

    public static Registry getInstance()
    {
        return soleInstance;
    }

    private Registry()
    {
    }

    ...
}
```

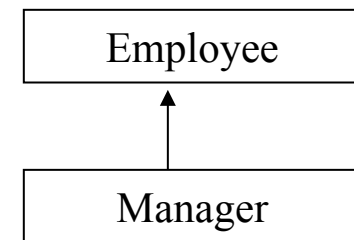
```
Registry registry = Registry.getInstance();
//registry = new Registry (); VIRKAR EKKI!!!
```

Erföir

# Erfðir í Java

- Klasar erfa aðra klasa
  - Object er rótin
  - Undirklasi (subclass) er klasi sem erfir annan klasa
  - final klasa er ekki hægt að erfa
- *extends* notað til að skilgreina erfðir
  - *super* vísar í erfðann klasa

```
class Manager extends Employee  
{ ... }
```



- *Polymorphic* eiginleikar
  - Tilvísunarbreytur geta vísað á undirklasa

```
Employee e;  
e = new Employee(. . .)  
e = new Manager(. . .)
```



# Dæmi

```
public class Employee
{
    private String name;
    private double salary;
    private Date hireDate;

    public Employee()
    {
    }

    public Employee(String name, double salary, Date hireDate)
    {
        this.name = name;
        this.salary = salary;
        this.hireDate = hireDate;
    }

    public Employee(String name)
    {
        this.name = name;
    }
}
```

# Dæmi

```
public String getName()
{
    return name;
}
...
public void setName(String name)
{
    this.name = name;
}
public String toString()
{
    return "Employee: " + getName();
}
}
```

# Dæmi

```
class Manager extends Employee
{
    String title;
    double bonus;
    public Manager (String name, String title)
    {
        super (name);
        this.title = title;
    }
    public String getTitle ()
    {
        return title;
    }
    public double getSalary()
    {
        return this.bonus + super.getSalary();
    }
    public String toString ()
    {
        return "Manager: " + getName() + ", " + getTitle ();
    }
}
```

← **Nýjar breytur**

← **Nýtt fall**

← **Yfirskrifað fall**

# Dæmi

```
public class Test2
{
    public static void main (String[] args)
    {
        System.out.println ("Test2");
        Test2 test2 = new Test2();
    }

    public Test2 ()
    {
        Employee e0 = new Employee ("Dilbert");
        Employee e1 = new Manager ("Pointy Haired", "Boss");
        System.out.println("e0: " + e0);
        System.out.println("e1: " + e1);
    }
}
```

```
C:\java>java Test2
Test2
e0: Employee: Dilbert
e1: Manager: Pointy Haired, Boss
```

# Dæmi með array

```
public Test2 ()
{
    Employee e0 = new Employee ("Dilbert");
    Employee e1 = new Manager ("Pointy Haired", "Boss");

    Employee elist[] = new Employee[2];
    elist[0] = e0;
    elist[1] = e1;

    for (int i=0;i<2;i++)
    {
        System.out.println(i + ":" + elist[i]);
    }
}
```

```
C:\java>java Test2
0:Employee: Dilbert
1:Manager: Pointy Haired, Boss
```

# Dæmi með tagbreytingu

```
public Test2 ()
{
    Employee e0 = new Employee ("Dilbert");
    Employee e1 = new Manager ("Pointy Haired", "Boss");

    e1.getTitle (); virkar ekki!

    for (int i=0;i<2;i++)
    {
        System.out.println(elist[i].getName());
        if (elist[i] instanceof Manager)
        {
            Manager m = (Manager)elist[i];
            System.out.println(m.getTitle ());
        }
    }
}
```

```
Dilbert
Pointy Haired
Boss
```

# Dynamic binding

- Java nota “dynamic binding”
  - Ákvörðun um hvaða fall er keyrt er tekin á keyrslu tíma
  - Sýndarvélin býr til *method table* fyrir hvern klasa

```
Manager m = new Manager();  
m.setName("P.H. Carl"); // fallið Employee.setName er keyrt  
m.setTitle ("Boss");    // fallið Manager.setTitle er keyrt  
m.getSalary ();         // fallið Manager.getSalary er keyrt
```

```
Employee e1 = new Manager("Pointy Haired", "Boss");  
e1.getSalary();
```

- Kostir *dynamic binding*
  - Ekki þarf að þýða grunnklasa ef nýjum klasa er bætt við

# Abstract klasar

- Abstract klasar gefa kost á því að láta undirklasa útfæra föll
  - `abstract` föll
- Þarf ekki að gera öll föll *abstract*
- Ekki er hægt að búa til tilvik af abstract klasa
- Hægt er að nota *abstract* tilvísunartög
  - Tilvik verður að vera venjulegur (concrete) klasi
- Klasar sem erfa *abstract* klasa verða að útfæra öll *abstract* föll
  - Annars eru þeir líka *abstract*



# Abstract dæmi

```
abstract class Person
{
    private String name;
    public Person(String name)
    {
        this.name = name;
    }
    // get og set föll ...
    public abstract String getDescription ();
}
class Employee extends Person
{
    public String getDescription()
    {
        return "Employee called " + super.getName();
    }
}

// Person p1 = new Person (); Virkar ekki!
Person p2;
Person p3 = new Employee ("Dilbert");
System.out.println (p3.getDescription());
```

# Generic Programming

- Allir klasar erfa frá *Object*
  - Getum búið til algorithmna og gagnagrindur sem eru generískar

```
static int find (Object[] a, Object key)
{
    int i;
    for (i=0;i<a.length;i++)
        if (a[i].equals(key)) return i;
    return -1;
}
```

```
Employee[] staff = new Employee[10];
Employee e1 = new Employee("Dilbert");
...
int n = find(staff, e1);
```

# Generic Programming

- Generískar gagnagrindur
  - *ArrayList* er dæmi um gagnagrind sem notar *Object*

```
ArrayList al = new ArrayList();  
al.add (new Employee ("Dilbert"));  
al.add (new Employee ("Wally"));  
al.add (new Employee ("Alice"));
```

```
Iterator i = al.iterator();  
Employee e;  
while (i.hasNext())  
{  
    e = (Employee)i.next();  
    System.out.println(e.getName());  
}
```

Dilbert  
Wally  
Alice

Skil

# Skil - Interface

- Skil eru klasar án útfærslu
  - Segja hvernig eigi að útfæra klasa
- Klasar geta útfært skil
  - *implements* skipunin
- Klasar sem útfæra skil verða að hafa þau föll sem talin eru upp í skilunum
- Eiginleikar skila
  - Öll föll og breytur eru *static final*
  - Skil hafa ekki útfærslu á klösum

# Dæmi um skil

```
public interface Comparable
{
    public int compareTo(Object other);
}
```

```
class Employee extends Person implements Comparable
{
    public int compareTo(Object o)
    {
        Employee e = (Employee)o;
        return this.getName().compareTo (e.getName());
    }
    ...
}
```

```
Employee[] ale = new Employee[3];
ale[0] = new Employee ("Dilbert");
ale[1] = new Employee ("Wally");
ale[2] = new Employee ("Alice");

Arrays.sort(ale);
for (int j=0; j < ale.length; j++)
    System.out.println(ale[j].getName());
```

```
Alice
Dilbert
Wally
```

# Notkun skila

- Ekki er hægt að búa til tilvik af skilum
- Hægt er að búa skil sem tilvísunartög
  - Verða að benda á tilvik sem útfæra skilin

```
Comarable c = new Comparable (); // VIRKAR EKKI!
```

```
Comarable c = new Employee (); // OK!
```

- Skil geta erft önnur skil
- Skil geta haft fasta

```
public interface Powered extends Movable  
{  
    double milesPerGallon();  
    double SPEED_LIMIT = 95;  
}
```

# Dæmi: teiknikerfi

```
public interface Drawable
{
    public void draw ();
}
```

```
public class Rectangle implements Drawable
{
    private int x, y, h, w;
    public Rectangle (int x, int y, int h, int w)
    {
        this.x=x; this.y=y; this.h=h; this.w=w;
    }
    public void draw ()
    {
        System.out.println ("(" +x+" "+y+" "+h+" "+w+"")");
    }
}
```



## Dæmi: teiknikerfi

- Kóði sem teiknar teiknihluti
  - Allir teiknihlutir útfæra *Drawable* og því notum við *draw* fallið

```
Iterator i = al.iterator();
Drawable d;
while(i.hasNext())
{
    d = (Drawable)i.next();
    d.draw();
}
```

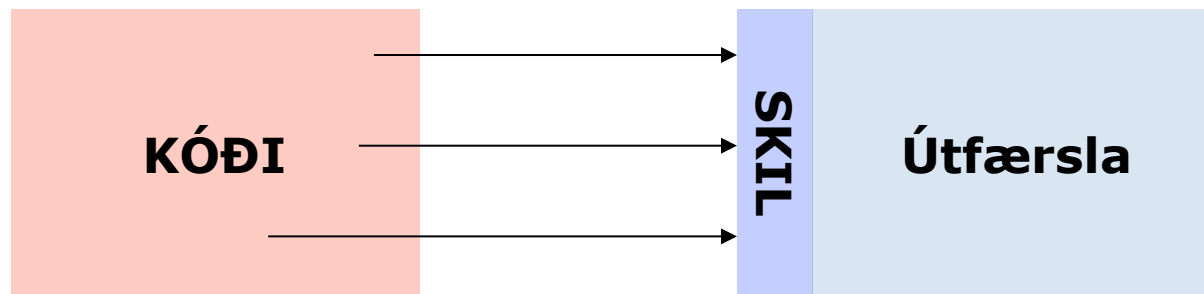
## Afhverju skil?

- Er ekki nóg að vera með abstract klasa?
- Abstract klasar duga ekki alltaf því við getum aðeins erft einn klasa
- Hægt er að útfæra mörg skil
- Skil falla vel að hönnunarhugmyndum
- Skil fela útfærslu
  - Klasar sem nota skil verða óháðir ákveðinni útfærslu

```
class Employee extends Person implements Comparable  
{  
    ...  
}
```

# Dæmi um skil

- *Table Data Gateway* mynda skil við gagnagrunn
  - Getum ákveðið við uppsetningu hvaða útfærslu skuli nota án þess að breyta kóða



- Dæmi

```
public interface TeamDAO extends FwDAO
{
    public void addTeam (Team team);
    public Collection getTeams ();
}
```

Undantekningar

# Undantekningar

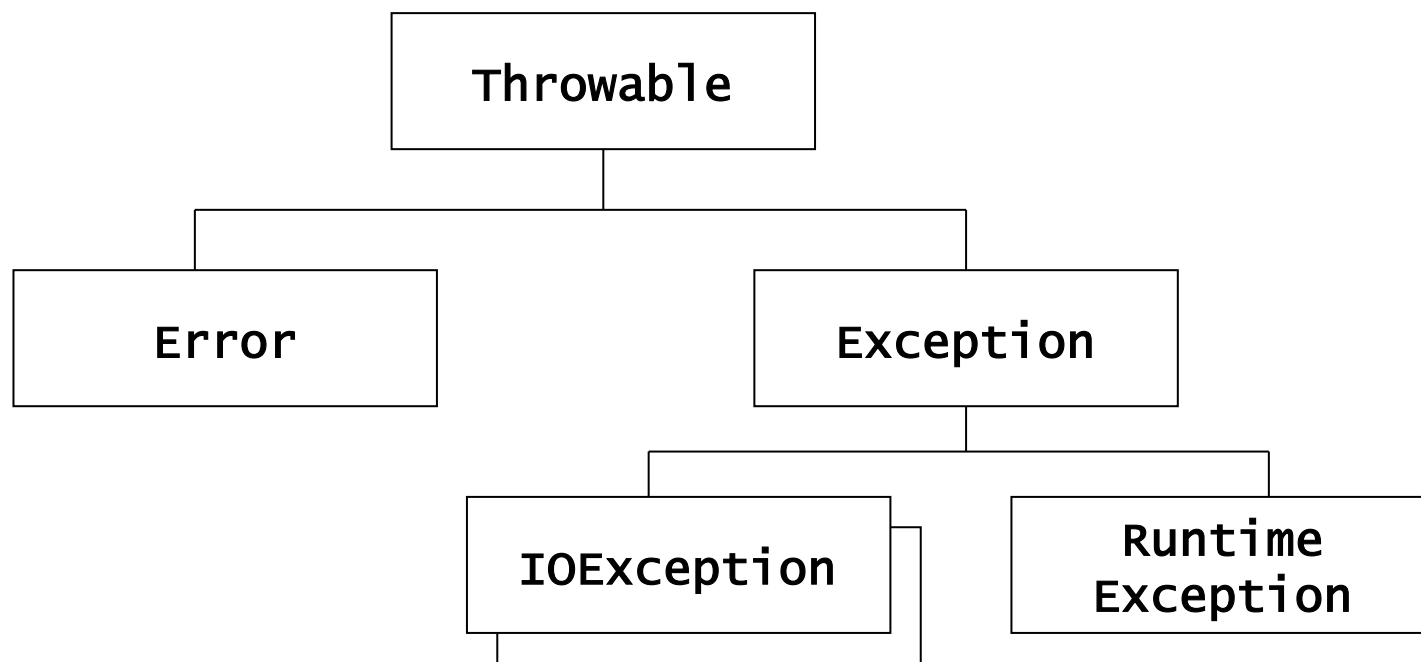
- Villumeðhöndun í Java byggir á undantekningum
  - Forritun “best case” er einföld
  - Forritun er ekki lokið fyrr en tekið er á villum
- Villumeðhöndlun er lykilatriði í forritun
  - Stór hluti af kerfisforritun
- Dæmi
  - Inntak er rangt
  - Skrá finnst ekki
  - Netið er niðri, gagnagrunnur finnst ekki

# Undantekningar

- Java útfærir undantekningar (*exceptions*)
    - Skilgreinir *try*, *catch* og *finally* í málinu
    - Klasinn *Exception*
  - *finally* er alltaf keyrt
  - *throw* kastar undantekningu
- ```
try  
{...}  
catch (Exception e)  
{...}  
finally  
{...}
```
- ```
throw new Undantekning ();
```
- Eitt *catch* fyrir hverja undantekningu
  - Ef fall grípur ekki undantekningar heldur kastið áfram
    - Endar í sýndarvélinni

# Undantekningar

- Undantekningar erfa frá *Throwable*
  - Klasarnir *Error* og *Exception*
  - *Error* er alvarleg villa – notað af sýndarvélinni
  - *Exception* er hægt að meðhöndla



# Undantekningar

- Ómerktar undantekningar (unchecked)
  - Geta alltaf komið upp
  - Dæmi: *OutOfMemoryError*, *NullPointerException*
- Merktar (checked)
  - Sérstaklega tiltekna, verður að grípa

```
public static String readFirstLine(String filename)
    throws IOException
{ ...
```



# Undantekningar

```
class FileNotFoundException extends Exception
{
    public FileNotFoundException (String s)
    {
        super(s);
    }
}
```

```
class FileManager
{
    void skrifaTilvik (Object o) throws FileNotFoundException
    {
        if (o == null)
            throw new FileNotFoundException ("Tilvikið er null!");
    }
}
```

```
FileManager fm = new FileManager();
try {
    fm.skrifaTilvik (null);
}
catch (FileNotFoundException fex) {
    System.out.println(fex);
}
```

```
C:\java>java File
FileNotFoundException: Tilvikið er null!
```

# Undantekningar

- Getum kastað undartekningum áfram

```
public static void main (String args[])
    throws MalformedURLException, IOException
{
    URL url = new URL ("http://www.ru.is");
    BufferedReader in = new BufferedReader(
        new InputStreamReader(url.openStream()));
    while (in.ready())
    {
        System.out.println (in.readLine());
    }
}
```

# Meðhöndlun undantekninga

- Undantekningar eiga ekki að vera eðlilegt flæði í forriti
- Ef undantekning verður
  - Kasta áfram, láta notanda klasans fá vandann
  - Grípa og breyta í *application-specific* undantekningu sem meðhöndluð er á efri lögum
- Meðhöndlun
  - Mikilvægt að geta rakið villur
  - Logga villuna
  - Undantekningar þurfa að innihalda réttar og nákvæmar upplýsingar

# Efnistöð

- Stuttur inngangur um Java
- Tilvísunartög
- Hlutbundin forritun
- Klasar
- Erfðir
- Skil
- Undantekningar