Java gyorstalpaló

Ekler Péter - peter.ekler@aut.bme.hu

Hudák Péter - hudakpe@gmail.com

Dér Kristóf - kristof.der@schdesign.hu

Gerencsér Péter – peter.gerencser@aut.bme.hu

BME AUT

AMORG APPLIED MOBILE RESEARCH GROUP

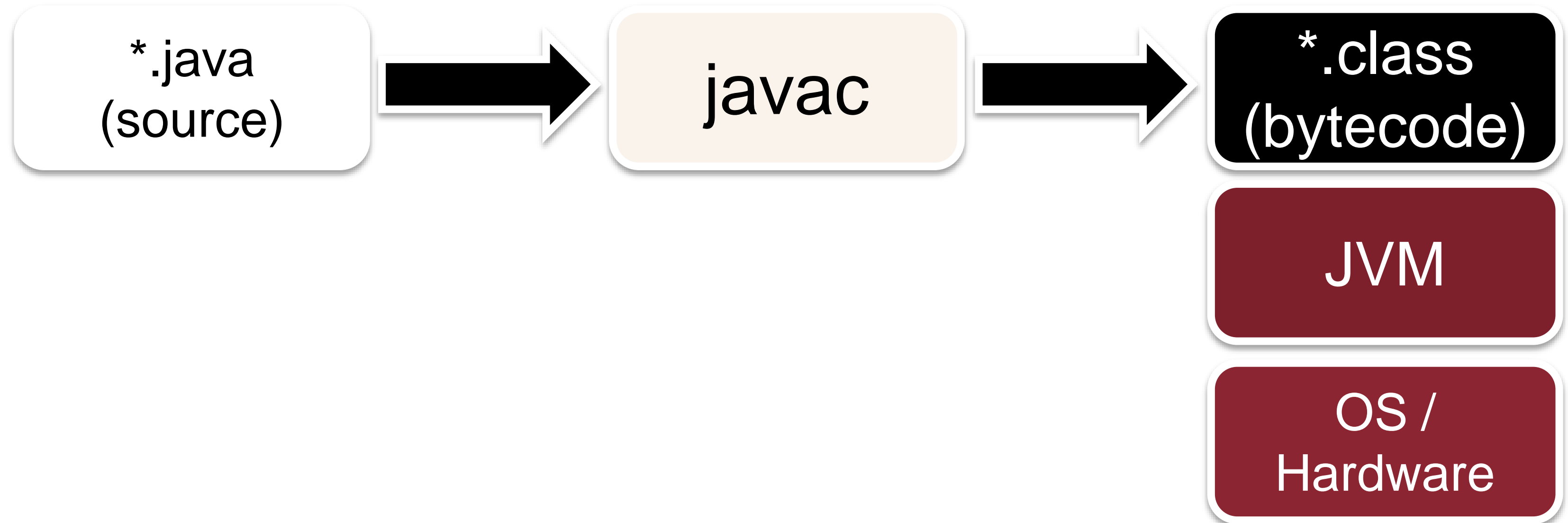
Tartalom

- Java alapok
- Osztályok, objektumok
- Interfészek, abstract osztályok
- Láthatóság
- Primitív típusok, operátorok
- Kivétel kezelés
- Konkurencia kezelés

Java Platform

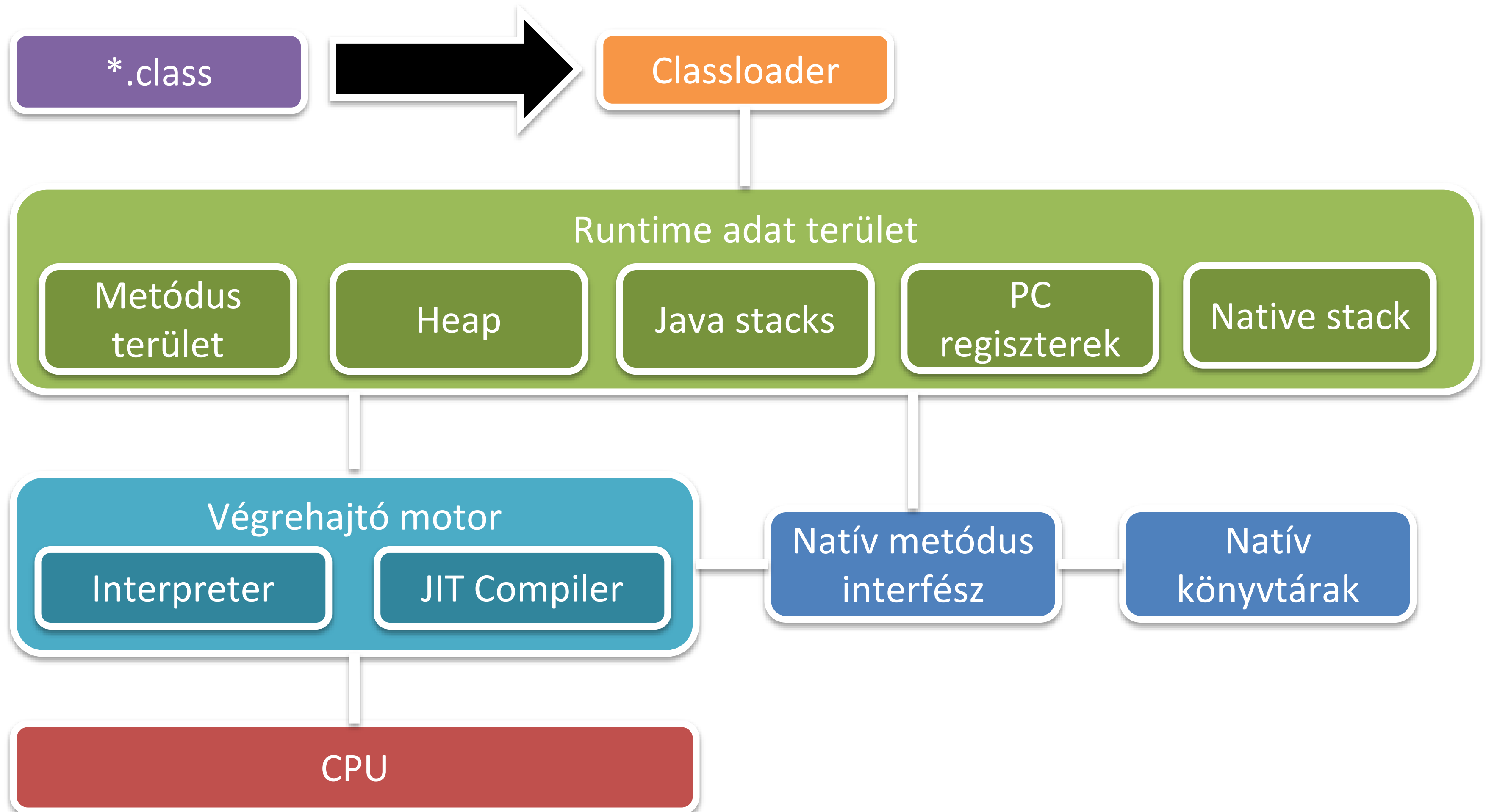
- JVM (Java Virtual Machine)
  - Hardver és operációs rendszer függetlenséget biztosít
- Java API
  - Alap funkciók
  - Kollekciók, I/O, hálózatkezelés, XML kezelés, konkurencia, stb.

Java fordítás



- JDK (Java Development Kit), JRE (Java Runtime Environment)
- Legfontosabb parancssori parancsok:
  - java, javac, javadoc, jar

JVM Architektúra



Java nyelv

- Objektum orientált
- Többszálúság támogatása
- C++-hoz hasonló
- Automatikus memória kezelés
- Nincs direkt hozzáférés a pointerekhez

Primitív típusok

Type	Size	Default (Field)
byte	8 bit	0
short	16 bit	0
int	32 bit	0
long	64 bit	0L
float	32 bit	0.0f
double	64 bit	0.0fd
char	16 bit (Unicode)	'\u0000'
boolean	Not specified	false

Operátorok

- Értékadás: = és annak kombinációi

- Aritmetikai operátorok: `+ - % / * ++ --`
- Relációs operátorok: `< > <= >= == !=`
- Logikai operátorok: `! && ||`
- Bit operátorok: `& | ^`
- `instanceof`

Operátor precedencia

Operátor	Precedencia
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Vezérlő struktúrák

- Return
- If-else

```
if (isConditionATrue())
    return 1;
} else if (conditionB && conditionC) {
    return 2;
} else {
    return 3;
}
```

- Inline if

```
int result = conditionA && conditionB ? 2 : 3;
```

Vezérlő struktúrák

- Switch: Java SE 6-ban csak int-re

```
int condition = 1;
switch (condition) {
    case 1: doSomething(); break
    case 2:
    case 3: doSomeOtherThing();
    case 4: doYetAnotherThing(); break;
    default: doDefaultThing();
}
```

Ciklusok

- For

```
for (int i = 0; i < 100; i++) {
    System.out.println(i);
}
```

- For each: tömbökön és kollekciókon való iteráció

```
for (String label : labelList) {
    System.out.println(label);
}
```

Ciklusok

- While

```
while (condition) {
    doSomething();
}
```

- Do-while

```
do {
    doSomething();
} while (condition);
```

Vezérlő struktúrák

- Return
- Break
- Continue

Alap OOP elvek

- Osztály: objektum terv, típus
- Objektum: állapot és viselkedés
- Encapsulation: egységbe zárás, hozzáférés a tagokhoz
- Öröklés: osztály hierarchia, általánosítás, specializálódás
- Interfész: „megegyezés” a külső világ felé

Hello World

```
class HelloWorldApplication {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Csomagok

- Névtér jellegű funkcionalitás Java-ban
- Csoportosítás és hozzáférés
- Teljes név használata:
  `com.example.ExampleClass`
- Konvenció:
  - Csomagnevek kisbetűsek
  - Osztálynevek nagybetűsek

Osztályok

Osztály deklaráció és csomag elnevezés

`./hu/bme/aut/amorg/demoproject/DemoClass.java:`

```
package hu.bme.aut.amorg.demoproject;
    public class DemoClass{
}
```

Osztályok

Láthatóság:

- public: minden más osztályból elérhető
- package private: csak a csomagon belüli osztályokból érhető el

Mezők

- Osztály tagváltozók állapot tárolásához
- Mező leírók:
  - Opcionális láthatósági paraméter
  - Típus
  - Név
  - Opcionális kezdő érték
- Az osztály elején szokás a mező deklarációt elhelyezni

Mezők

Mező deklarációs példa

```
package hu.bme.aut.amorg.demoproject.field;

import hu.bme.aut.amorg.demoproject.class.*;

public class FieldDeclarationExample {
    public ClassDeclarationExample publicField;
    private long privateField = 2L;
    byte packagePrivateField = 3;
}
```

Mezők

Láthatóság beállítása:

- *Nincs*: package private, a mező az osztály csomagján belül érhető el
- *Private*: a mező csak az osztályában érhető el (öröklő nem éri el)
- *Protected*: a mező csak a package-ből és a leszármazott osztályokból érhető el
- *Public*: a mező bárhonnan elérhető

Metódusok

- Metódus deklaráció:
  - Opcionális láthatóság
  - Visszatérési típus
  - Függvény név
  - Paraméter lista zárójelben
  - Opcionális „throws” deklaráció
  - Metódus törzs

Metódusok

- Metódus szignatúra: név és paraméter lista
- Egy osztályban minden metódusnak egyedi szignatúrával kell rendelkeznie
- Nincsenek metódus pointerek

Metódusok

Metódus deklaráció példa:

```
public class MyClass{

    private long height = 2L;

    public int getBackValue(int returnValue) {
            return returnValue;
    }
    void modifyHeight() {
            height = 4L;
    }
}
```

Metódus láthatóság

- *Nincs*: a package osztályain belül érhető el a metódus
- *Private*: a metódus csak az osztályából érhető el (öröklő nem éri el)
- *Protected*: a metódus csomag osztályaiból és a leszármazott osztályokból érhető el
- *Public*: bárhonnan elérhető

Clean Code elvek - metódus

- Legyenek rövidek a metódusok (5-10 sor)
- Egy metódus egy feladatot lásson el
- Az absztrakció sorrendjében rendezzük a metódusokat
- Kerüljük a háromnál több paramétert
- Ne legyenek mellékhatásai a visszatérési értékeknek
- Ne adjunk át és ne térjünk vissza `null`-lal
- Ne legyen ismétlés

Clean Code elvek - osztály

- Az osztályok kicsik legyenek, egy dologért legyenek felelősek
  - Kivétel: Util osztályok (log, convert, formázás)
- Tervezzünk a változó igényekhez igazodva
- Minimalizáljuk az osztály mezők és függvények láthatóságát
- Getter/Setter használata public elérhetőség helyett mezők számára

Öröklés

- A Java csak egyszeres öröklést támogat, de bármennyi interfészt implementálhatunk
- Minden osztály az Object-ből származik le
- Az alosztály örökli az ős osztály minden publikus és protected mezőjét és metódusát (illetve package-n belül a package-private-okat is)
- Leszármazott osztályban a super hívás használható az ős elérésére
- A final kulcsszó használható osztály és metódus esetén, hogy a felüldefiniálást megtiltsuk

Öröklés példa

```
public class Animal {
        //Object az őse
        private int age = 3;
        public int getAge() {
                return age;
        }
}

public class Dog extends Animal {
        @Override
        public int getAge() {
                return age*7;
        }
}
...
Animal a = new Animal();
a.getAge();     // 3
Dog d = new Dog();
d.getAge();     // 21
```

Cast-olás

- Általánosítás támogatása
- Implicit upcasting (Liskov helyettesítés):

```
Object upcastedObject = new SpecializedObject();
upcasteObject.toString();
```

- Explicit downcasting:

```
SpecializedObject specializedObject =
(SpecializedObject)upcastedObject;
```

Interfészek

Példa:

```
public interface ExampleInterface {
    void interfaceMethod();
}

public interface ExtendedExampleInterface extends
ExampleInterface {
    void extendedInterfaceMethod();
}
```

Interfészek

Implementációs példa:

```
public class InterfaceImplementation implements
ExtendedExampleInterface {

    @Override
    void interfaceMethod() {
    }

    @Override
    void extendedInterfaceMethod() {
    }
}
```

Interfészek

- Az interface kulcsszóval deklarálható
- Csak konstansokat és metódus szignatúrát tartalmazhat
- Nem lehet példányosítani
- Osztályok implementálhatják
- Egy osztály több interfészt is implementálhat, de minden metódust felül kell definiálni
- Több interfészt is kiterjeszthet
- Alapértelmezetten minden metódus public és abstract, illetve minden mező konstans (public static final)

Abstract osztályok

- Abstract kulcsszóval deklarálható
- Nem lehet példányosítani de lehet leszármaztatni belőle
- Bármit tartalmazhat amit egy osztály
- Tartalmazhat abstract metódusokat

Objektumok

- Minden objektum
- Az objektum egy osztály példánya
- Az objektumok referencia szerint kerülnek átadásra a metódus hívásokban
- Ha nincs több referencia az objektumra, a memória fel lesz szabadítva a Garbage Collector által

Konstuktor

- „Metódus” az objektum inicializálására
- Deklaráció:
  - Opcionális láthatóság
  - Osztály neve, mint metódus név
  - Paraméterek
  - Opcionális throws deklaráció
  - Metódus törzs
- Ha nincs konstruktor, akkor a compiler az osztályhoz rendel egy paraméter nélküli default konstruktort

Konstuktor

- Egy osztálynak több is lehet
- A konstruktor hívhat más konstruktorokat is a *this()*-el, de csak a legelején
- Hívható az ősosztály konstruktora a super() hívással, de csak a legelején
- A this kulcsszó az aktuális példányra referál

Objektum életciklus

1. Memória foglalás az objektum számára
2. Mezők default értékeinek beállítása
3. Ősosztály konstruktor hívás (implicit v. explicit)
4. Példány változók és inicializáló blokkok az inicializálás sorrendjében
5. Konstruktor body

Statikus mezők és metódusok

- Osztály szintű változók és metódusok
- *static* kulcsszó
- A statikus metódusok csak statikus változókat és más statikus metódusokat érhetnek el
- A statikus metódusok nem polimorfak (statikus kötés)
- Leszármazott osztályok nem „módosíthatják” a statikus viselkedést
- Statikus kód akkor fut, amikor először nyúlunk az osztályhoz

A final kulcsszó

- Osztályoknál: öröklés tiltása
- Metódusoknál: felüldefiniálás tiltása
- Mezőknél és változóknál: csak egyszer állítható be
- Statikus változóknál: konstans lesz
- Metódus paramétereknél: paraméterek újra beállítását akadályozza meg

Inner class

- Osztály definíció osztályon belül
- public, private, protected és package private minősítők használhatók
- Használat:
  - Logikai csoportosítás
  - Egységbe zárás (adat struktúra, lokális implementáció)
  - Olvashatóság
- Példányosítás:

```
OuterClass.StaticNestedClass nestedObject = new
OuterClass.StaticNestedClass();
```

Inner class továbbiak

- Lokális inner class:
  - Metóduson belüli definíció
  - Nem tagja a külső osztálynak
  - Csak a külső osztály tagjaihoz és a final jelzőjű lokális változókhoz fér hozzá

```
public class Foo {

    private class Bar {
        //...
    }
}
```

Inner class továbbiak

- Anonymous inner class:
  - Név nélküli lokális inner class
  - A konstruktor nem, de az inicializáló blokk megadható

```
java.awt.EventQueue.invokeLater(new Runnable() {
         @Override
         public void run() {
            createAndShowUI();
         }
      });
```

Tömbök

- Tömbök is objektumok
- Van egy publikus `length` mezőjük
- Fix tartalommal, vagy mérettel kell inicializálni
- Alapból fel van töltve az adott típus alapértékével
  - Egyszerű típusoknál 0 (nem kell kinullázni!)
- [] operátorral és indexszel érjük el az elemeket
- Ellenőrzik az indexet, dobhatnak `IndexOutOfBoundsException`-t
- Tartalmazhatnak primitív típust, de nem tartalmazhatnak generikust
- java.util.Arrays ad néhány segédmetódust

Tömbök

Példák:

```
int[] measuredValues = new int[10];
int referenceValues[] = {1,2,3,4,5,6,7,8,9,10};
```

Változó számú argumentumlista

- Különbözik a szintaxis a tömb típusú argumentumtól
  - De a változó számú paraméterlistát tömbként kapjuk meg
- Bizonyos esetben olvashatóbb kódot eredményez
- Csak utolsó paraméterként használható
- Példa:

```
…
public int sum(int… values) {
          int result = 0;
          for (int value : values) {
                    result += value;
          }
          return result;
}
…
sum(2,3,4);
sum();
```

Kivételkezelés

- Segítenek kezelni a hibákat és a váratlan eseményeket
- Dobni egy kivételt annyit jelent, mint példányosítani egy `Exception`-t és átadni a JVM-nek a `throw` kulcsszóval
- A JVM átadja a legközelebbi kivételkezelőnek - `catch`

Kivételkezelés

Kivétel típusok:

- Runtime exceptions – általában programozói hibák (pl. NullPointerException)
- Errors – alkalmazáson kívüli hibák (pl. OutOfMemoryError)
- Checked exceptions – minden más

Kivételkezelés

Throwable

Exception

Error

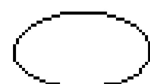
RuntimeException

any class besides RuntimeException

any subclass

any subclass

any subclass

 A checked throwable

An unchecked throwable

Kivételkezelés

- Metódusdeklaráció tartalmazhatja a `throws` kulcsszót. Utána kell felsorolni a kiváltható Exception-ök listáját.
- Az ellenőrzött kivételeket kötelező elkapni vagy deklarálni a metódus elején, hogy dobhat olyat.

Kivételkezelés

Példa:

```
public class ExceptionExample {
        private FileInputStream openFile() throws
                FileNotFoundException, IOException {
        …
        }
        public void processFile() {
                try {
                        openFile();
                } catch (FileNotFoundException fe) {
                        System.out.println("File not found!");
                } catch (Exception e) {
                        System.out.println("Error occured!");
                        e.printStackTrace();
                }
        }
}
```

Kivételkezelés

- A `catch` blokk után egy `finally` blokkot is fel lehet venni:
  - Mindig lefut (majdnem mindig, JVM-ből kilépés és szál lelövése kivétel)
  - Erőforrás felszabadításra használható
- A kivételeket lehet láncolni (továbbdobni)
- Exception-ből örökléssel saját kivétel osztály hozható létre

Kivételkezelés

Előnyök:

- Hibakezelő kód elkülönül
- Call stack terjesztés
  - Az exception tartalmazza a dobáskori call stack-et
- Hiba típusok szeparálása
- Nehezebb figyelmen kívül hagyni

Kivételek és Clean Code

- A hibakezelés csak egy dolog
- Használj kivételeket a hibakódok visszatérése helyett
- Használj nem ellenőrzött kivételt a programozási hibákra
- Használj ellenőrzött kivételt a hibákra, amikre fel lehet készülni
- Használj saját kivételeket, hogy összefogd és továbbítsd a hibákat magasabb szintekre
- Lehetőség szerint használj beépített kivételeket
- Absztrakciós szintnek megfelelő kivételt használj

Kivételek és Clean Code

- Ne vidd túlzásba a kivételeket!
- Ne hagyd figyelmen kívül a kivételeket!
- Ne kapj el Throwable-t, ha nem vagy teljesen biztos a dolgodban

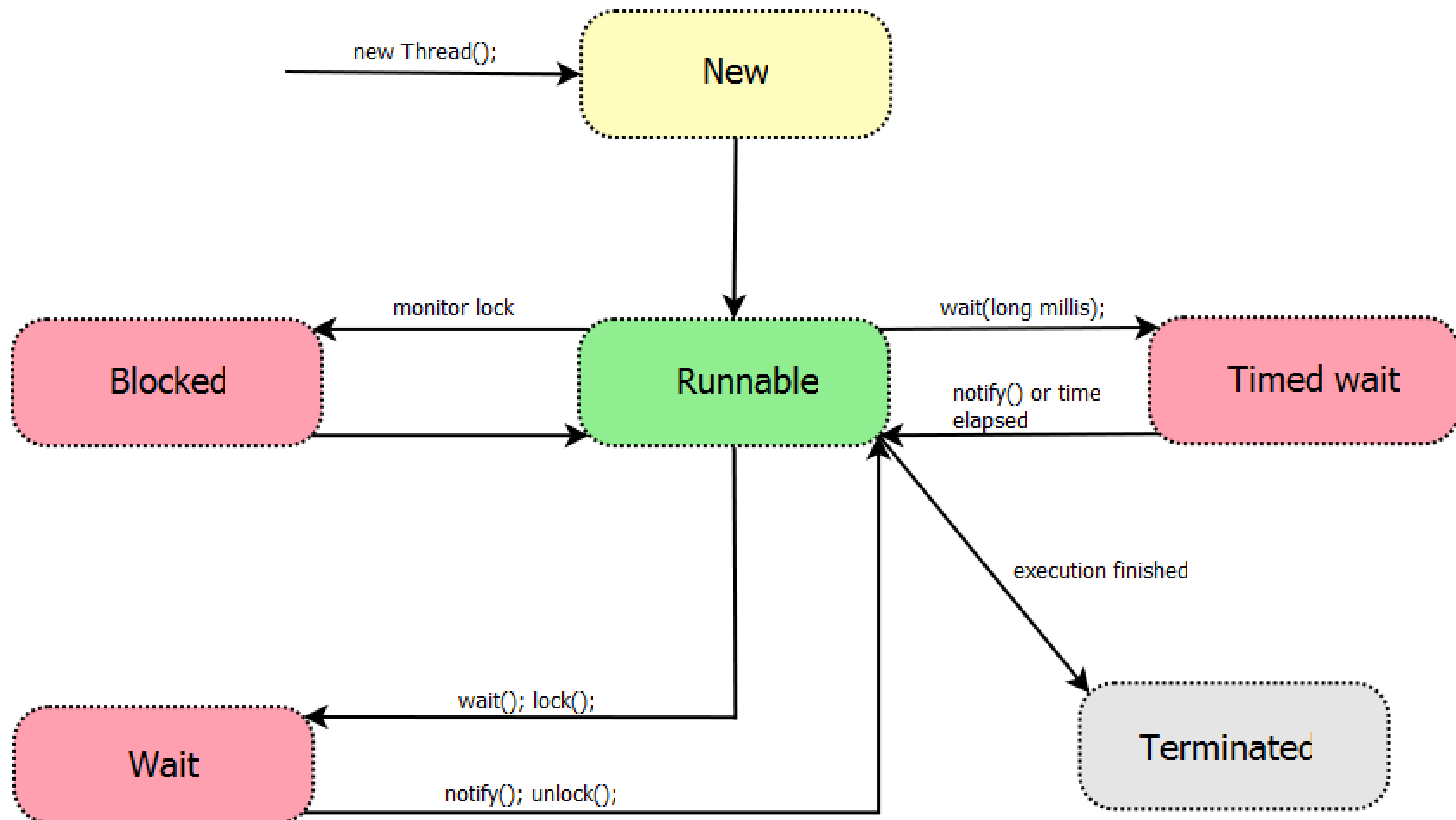
Szálak

- Blokkolódó műveletek esetén gyakran használatos!

```
class MyJob extends Thread {
  public void run()
  {
    // feladat végrehajtása akár ciklusban…
  }
}

// meghívás kívülről
new MyJob(/*esetleges
 paraméterek*/).start();
```

Szál állapotok



Szálak létrehozása

- Kétféle létrehozás

```
public class ThreadTest extends Thread {

    @Override
    public void run() {
        // do something
    }

    public static void main(String[] args) {
        ThreadTest  test = new ThreadTest ();
        test.start();
    }

}
```

```
public class ThreadTest {

    public static class MRun implements Runnable {
        @Override
        public void run() {
            // do something
        }
    }
    public static void main(String[] args) {
        Thread  test= new Thread (new MRun());
        test.start();
    }

}
```

Szál példa

```
public class Main {

  public static lock = new Object();

// ...

 public static void main(String arg[]) {
   Thread1 thread1 = new Thread1();
   Thread2 thread2 = new Thread2();

   thread1.start();
   thread2.start();

    int i = 0;
    i++;
   try {
     thread1.join(1);
     thread1.interrupt();
     thread2.join();
   } catch (InterruptedException ex) {
  }
}
```

```
// ...

public void run() {

int i = 0;
i++;

synchronized (Main.lock) {
   try {
        Thread.sleep(100);
    } catch (InterruptedException ex) {
    }
    Main.lock.wait();
  }

}

// ...
```

```
// ...

public void run() {

int i = 0;
i++;

synchronized (Main.lock) {
     Main.lock.notify();
  }

}

// ...
```

SOLID elvek

Single responsibility principle:

Egy objektum egy dologért felel

Open/closed principle:

Az entitások nyitottak a kiterjesztésre de zártak a módosításra

Liskov substitution principle:

Az objektumok helyettesíthetők altípusokkal

Interface segregation principle:

Többszörös és speciális kliens interfészek

Dependency inversion principle:

A környezet igazodik a magas szintű modul interfészhez

Elnevezési konvenciók

- Osztályok, abstract osztályok, interfészek:
  - Minden szó első betűje nagybetű
- Mezők:
  - Minden szó nagybetűvel kezdődik, kivéve az első ami kisbetűs
- Metódusok:
  - Minden szó nagybetűvel kezdődik, kivéve az első ami kisbetűs
  - Az első szó ige
- Konstans:
  - Nagybetűs, szavak határán aláhúzás jel

Clean Code elvek - elnevezés

- Vegyük komolyan!
- Kiejthető, jól kereshető és jól leíró neveket használjuk
- Kerüljük a félre tájékoztató neveket
- Soha sem csak magunknak írjuk a kódot
- Nem kell feltétlenül félni a hosszú nevekből
- Az osztálynevek főnevek legyenek (kerüljük az általános neveket mint Manager, Processor, Data, Info, stb.)
- Metódus név igével kezdődjön

Clean Code elvek - komment

- A kommentet főként JavaDoc generálására használjuk
- A felesleges komment csak zavaró
- A bonyolult algoritmusok érthetőségét lehet kommenttel segíteni
- Egy jól olvasható kódban nincs szükség kommentre

Javasolt források

- Oracle java tutorial:
  http://docs.oracle.com/javase/tutorial/
- Oracle Java SE 6 documentation:
  http://docs.oracle.com/javase/6/docs/index.html
- Bruce Eckel - Thinking in Java 4th Edition:
  Good for learning Java, intermediate level chapters for almost all Java topics
- Joshua Bloch - Effective Java 2nd Edition:
  Good for using Java, contains advices and best practices for specific topics
- Robert C. Martin - Clean Code:
  Recommended for everyone, Java is only used as an example language

Kérdések