**” Thinking in Java”, Bruce Eckel**

**Contents**

**Initialization & Cleanup 2**

**Reusing Classes 2**

**Polymorphism 8**

**Interfaces 9**

**Inner classes 13**

**Containers-Collections-Iterators 20**

**Exceptions/Error Handling 24**

**Strings/Regular Expressions/regex classes & methods 31**

**Type Information-RTTI-Reflection 33**

**Generics 37**

**Arrays 45**

**Initialization & Cleanup**

**Ex-17,18**

class Kls{

Kls(String sn){

System.out.println(sn); } }

public class Main { public static void main(String[] args) {

// Kls[] ar = new Kls[7]; For Ex. 17

Kls[] ar= { //Initialization for ex. 18

new Kls("sept"),

new Kls("sime"),

new Kls("cinq"),

new Kls("quatre"),

new Kls("trois"),

new Kls("deux"),

new Kls("un"),

};

**Ex. 19** static void fun(String...args){

for(String su:args)

System.out.println(su);

//3rd Method

String h= "eoa";

String fj= "bnnk";

String[] k = {h,fj};

fun(k);

/\* 2nd Method

String h= "eoa";

String fj= "bnnk";

String helma;

String[] k = new String[]{

h = "qpeor",

fj= "apooow",

helma= "apaoid",}; \*/

// fun(k);

/\* 1st Method:

String h= "eoa";

String fj= "bnnk";

fun(h,fj);\*/

**Ex. 20** public static void main(String... args) {

String h = "epoqqie";

String fj =" ancmq";

A.main( new String[] {h, fj, "7j8a",}); }

class A {

public static void main (String... args){

for (String s: args)

System.out.println(s); }

**Reusing classes Reusing classes Reusing classes**

**Ex. 1**

class Barbounia {

private String sin;

Barbounia() { sin = "sinijo";}

public String toString() {return sin; }

}

public class Main

{ private Barbounia igtis;

//public Main() {}

public String toString() {

if (igtis==null)

igtis = new Barbounia();

return

"Barbounia = " + igtis;

}

public static void main(String[] args) {

Main mina = new Main();

System.out.println(mina);

}

}

**Ex. 2** class Detergente {

private String sin= "Detergente\n";

public void append(String sula) { sin += sula; }

public void scrub() {append(" scrub() \n"); }

public String toString() {return sin; }

public static void main(String[] args) {

Detergente lisot = new Detergente();

System.out.println(lisot);

}

}

public class Main extends Detergente

{ public void scrub () { append("Disinfenctant scrub()\n"); }

public void sterilize(int k) {

switch (k) {

case 1: append("Sterilize with gamma rays!\n");

break;

case 2: append("Sterilize with sodium hypochlorate!\n");

break;

case 3: append("Sterilize with alcohol!\n");

break;

default: System.out.println("Gebe 1,2 oder 3 Eintrag ein!");

}

}

public static void main(String[] args) {

Main mina = new Main();

mina.scrub();

mina.sterilize(2);

System.out.println(mina);

}

}

**Ex. 4** class umo {umo() {System.out.println("umo");} }

class dos extends umo { dos() {System.out.println("dos");}}

class tres extends dos { tres() {System.out.println("tres");} }

public class Main extends tres

{ public static void main(String[] args) {

Main artsel = new Main();

}

}

**Ex. 5** class A {A() {System.out.println("A Konstruktor");} }

class B { B() {System.out.println("B Konstruktor");}}

class Cita extends A {

B vita = new B();

}

public class Main

{ public static void main(String[] args) {

Cita artsel = new Cita();

}

}

**Ex. 7** class A {A(int u) {System.out.println("A Konstruktor");} }

class B { B(double mes) {System.out.println("B Konstruktor");}}

class Cita extends A {

Cita(){ super(34); }

B vita = new B(-2);

}

public class Main

{

public static void main(String[] args) {

Cita artsel = new Cita();

}

}

**Ex. 8** class A {A(int u) {System.out.println("A Konstruktor" + " " + u);} }

class Cita extends A {

Cita(){ super(9); }

Cita(char mie){ super(-7); }

}

public class Main

{

public static void main(String[] args) {

Cita artsel = new Cita('q');

}

}

**Ex. 9, 10** class Wuerzel {

Wuerzel() { System.out.println(" Wuerzel Kons"); }

Inhalt\_1 In\_1 = new Inhalt\_1(-2);

Inhalt\_3 In\_3 = new Inhalt\_3(7);

}

class Stem extends Wuerzel {

Stem(char si) { System.out.println(" Stem Kons"); }

Inhalt\_3 In\_3 = new Inhalt\_3(21);

Inhalt\_1 In\_1 = new Inhalt\_1(-3);

}

class Inhalt\_1 {

Inhalt\_1(int tur) { System.out.println(" Inhalt\_1 Kons " + tur); }

}

class Inhalt\_3 {

Inhalt\_3(int ur) {System.out.println(" Inhalt\_3 Kons " + ur); }

}

public class Main

{

public static void main(String[] args) {

Stem artsel = new Stem('q');

}

}

**Ex. 11** class Cleanser {

private String s;

public void append(String a) { this.s = s;

s += a;

System.out.println(s);

}

public void apply() { append(" apply()"); }

}

public class Main extends Cleanser

{ Cleanser Silene = new Cleanser();

public void append(String a) { Silene.append(a); }

public void apply() { Silene.apply(); }

public static void main(String[] args) {

Main Deteren = new Main();

Deteren.append("");

Deteren.apply();

}

**Ex. 13**

class Varta {

void Juri(float f) { System.out.println("#"); }

void Juri(char omega) { System.out.println("@ W"); }

void Juri(double shiu) { System.out.println("\*"); }

}

public class Main extends Varta {

void Juri(String siun) {

System.out.println("Binnen Main");

}

public static void main(String[] args) {

Main lam = new Main();

lam.Juri("f");

lam.Juri(71);

lam.Juri('W');

} }

**Ex. 15** package paketo;

class Varta {

protected void Juri(char omega) { System.out.println("@ W"); }

}

public class Main extends Varta {

public static void main(String[] args) {

Main lam = new Main();

lam.Juri('W');

}

**Ex. 16, 17**

class Amfibiano {

protected char Roar() { System.out.println("Roar");

return 'R';}

protected char Paddle() {System.out.println("Paddle");

return 'P';}

}

class Fro extends Amfibiano {

protected char Roar() { System.out.println("Fro Style Roar!");

return 'R';}

protected char Paddle() {System.out.println("Fro Style Paddle");

return 'P';}

}

**Ex. 18** class Fro {

static final double tym= 574;

final double jea ;

public Fro(double i) {this.jea = i;}

public void Feld\_Kontroll() {

System.out.println(tym);

System.out.println(jea);

}

}

public class Main {

public static void main(String[] args) {

Fro lam = new Fro(89);

lam.Feld\_Kontroll();

//lam.jea = 27; // Illegal assignement. Constant field.

// lam.tym = 0; //Illegal assignement. Constant field and unique field for the class.

Fro san = new Fro(51);

san.Feld\_Kontroll();

}

**Ex. 19** class A { void R() {System.out.println("Innen A!"); }

}

class Fro {

final A ju ;

public Fro(A fin) {this.ju = fin; }

}

public class Main {

public static void main(String[] args) {

Fro lam = new Fro(new A());

lam.ju.R();

// lam.ju = new A(); //Error.Cannot assign

//a value to final variable ju.

}

**Ex. 21** class MitFinal {

final void f() { System.out.println("Mit Final"); }

}

class A extends MitFinal {

public void f() { System.out.println(" A A !"); }

}

public class Main {

public static void main(String[] args) {

A fan = new A();

fan.f(); //Error f() in A cannot override f in MitFinal.

**Revision**

class Anginara { static int i=0;

public Anginara() {++i; }

final void flora () { System.out.println("Das ist Anginara!"); }

void no() {System.out.println(i); }

}

class Anguri {static int i= 0;

public Anguri() { System.out.println("Anguri war beschafft!");

++i;}

void flora () { System.out.println("Das ist Anguri!"); }

void no() {System.out.println(i); }

}

class Herbivoro {

private String s;

public Herbivoro(String shium) { this.s = shium; }

Anginara ani = new Anginara(); //**Delegation**. Between inheritance

Anguri uri = new Anguri(); //...

void Essen() { ani.flora(); //...

uri.flora(); // and composition.

}

//public String toString() { return this.s +" isst Pflanzen... "; }

}

public class Main

{

public static void main(String[] args) {

Herbivoro Umus = new Herbivoro("Golparus");

Umus.Essen();

System.out.println(Umus);

Umus.ani.no();

Anguri suri =new Anguri();

suri.no();

}

}

E**x.** **22** final class Tia\_Maria {

} public class Main extends Tia\_Maria { /\* Error cannot inherit final class

Tia\_Maria.

**Ex. 23, 24**

class Insect {

private static int iu = 0;

protected int j;

Insect() {

iu++;

System.out.println(iu);

j = 39;

}

private static int x1 =

printInit("static Insect.x1 initialise");

static int printInit(String s) {

System.out.println(s);

return -7;

}

}

class Beetle extends Insect { static int iu=0;

Beetle() { ++iu;

System.out.println(iu);

}

}

class Kantaros extends Beetle {

}

public class Main extends Kantaros

{ private int k = printInit("Beetle.k initialise");

public static void main(String[] args) {

Kantaros Ka = new Kantaros();

new Beetle(); /\* The static fields will be incremented once when an

} an objects is created. The constructor is called once.

} Hence class loading occurs once per object creation.\*/

**Polymorphism Polymorphism Polymorphism**

**Ex. 9** class Rode {

protected void Essen() { System.out.println("Rode isst!"); }

protected void Hopfen() { System.out.println("Rode hoppft!"); }

}

class Mou extends Rode {

public void Essen() { System.out.println("Mou isst!"); }

public void Hopfen() { System.out.println("Mou hoppft!"); }

}

class Gerhard extends Rode {

public void Essen() { System.out.println("Gerhard isst!"); }

public void Hopfen() { System.out.println("Gerhard hoppft!"); }

}

public class Main

{

public static void main(String[] args) {

Rode[] Ohr = { new Mou(), new Gerhard() };

Ohr[0].Hopfen();

Ohr[1].Essen();

}

}

**Ex.17** class Zyklus {

}

class DreiZyklus extends Zyklus {

}

class MonoZyklus extends Zyklus {

public void Ebenheit() { System.out.println("Ebenheit erfuehrt!");

}

}

public class Main

{

public static void main(String[] args) {

Zyklus Dz = new DreiZyklus();

Zyklus Mz = new MonoZyklus();

Zyklus[] Zus = { Dz, Mz};

/\* Dz.Ebenheit(); Error Dz, Mz of type Zyklus.

Mz.Ebenheit(); Ebenheit exists only in MonoZyklus.

((MonoZyklus)Zus[0]).Ebenheit(); /\***Exception thrown DreiZyklus cannot be**

(due to previosu upcast) (down)**cast to MonoZyklus.** \*/

((MonoZyklus)Zus[1]).Ebenheit();

}

}

**Interfaces Interfaces Interfaces**

**Ex. 1** abstract class Rode {

protected void Essen() { System.out.println("Rode isst!"); }

protected abstract void Hopfen();

}

class Mou extends Rode {

public void Essen() { System.out.println("Mou isst!"); }

public void Hopfen() { System.out.println("Mou hoppft!"); }

}

class Gerhard extends Rode {

public void Essen() { System.out.println("Gerhard isst!"); }

public void Hopfen() { System.out.println("Gerhard hoppft!"); }

}

**Ex. 2** abstract class Rode {

protected void Essen() { System.out.println("Rode isst!"); }

protected abstract void Hopfen();

} public static void main(String[] args) {

Rode Ohr = new Rode(); // Error Rode is abstract

// cannot be instantiated.

**Ex. 3** abstract class Varta {

public Varta() { Print() ; }

abstract void Print();

}

class Bravo extends Varta {

private int suo=-71;

public void Print() { System.out.println(" suo = " +suo); }

}

public class Main

{

public static void main(String[] args) {

Bravo bot = new Bravo();

bot.Print(); // Prints: suo = 0

// suo = -71

While it is not possible to create an instance of the abstract base class when an object of the derived class

Is created the constructor of the base class is called which calls the print method of the derived class. The integer

Member-field has the default value of 0 hence it is printed equal to 0. Then the derived class constructor must be

Called which initializes the fields and the value of the field as defined in the body of the derived class is printed, -71.

This can be proved if the line below for the derived class constructor printing a message to declare its invocation is added

class Bravo extends Varta {

public Bravo() { System.out.println(" Bravo Konstruktor"); }

**Ex. 4** abstract class Varta {

}

class Bravo extends Varta {

private int suo =-71;

public void Print() { System.out.println(" suo = " +suo); }

}

public class Main

{ public static void Metodo(Varta vina) {

((Bravo)vina).Print(); // Downcasting

}

public static void main(String[] args) {

Varta vua = new Bravo(); // Upcasting

Metodo(vua);

**ii)** abstract class Varta {

public abstract void Print(); }...

public static void main(String[] args) {

Varta vua = new Bravo(); // Upcasting

vua.Print();

**Ex. 7** interface Rode {

void Essen();

void Hopfen();

}

class Gerhard implements Rode

**Ex. 9** abstract class Spielbar {

void spielen(Note n) { System.out.println("Spiel note n"); }

void einstellen() {System.out.println("Wurde eingestellt!");}

}

class Percussion extends Spielbar { }

public static void main(String[] args) {

Percussion rok = new Percussion();

tune(rok);

rok.einstellen();

**Ex. 10** interface Instru {

int VALUE = 5;

void einstellen();

} enum Note {

Middle\_C, C\_Sharp, B\_Flat;

}

interface Spielbar {

void spielen(Note n); //{ System.out.println("Spiel note n"); }

}

class Percussion implements Spielbar, Instru {

public void spielen(Note n) { System.out.println("Percussion Spiel note n"); }

public void einstellen() {System.out.println("Wurde eingestellt!");}

}

public class Main

{ static void tune(Spielbar i) {

i.spielen(Note.Middle\_C);

}

public static void main(String[] args) {

Spielbar rok = new Percussion();

tune(rok);

((Percussion)rok).einstellen();

**Ex. 14, 15** interface I1 {

void metodo1();

void metodo2();

}

interface I2 {

void metodo1();

void metodo2();

}

interface I3 {

void metodo1();

void metodo2();

}

interface I4 extends I1, I2, I3 {

void metodo1();

void metodo2();

void metodo3();

}

abstract class Alfa {

public void metodoaf(){System.out.println("metodo Alfa");}

}

class Bravo extends Alfa implements I4 {

public void metodo1(){ System.out.println("metodo1"); }

public void metodo2(){ System.out.println("metodo2"); }

public void metodo3(){ System.out.println("metodo3"); }

}

public class Main

{

public static void Meto1(I1 i) {

System.out.println("I1");

i.metodo1();

}

public static void Meto2(I2 i) {

System.out.println("I2");

i.metodo2();

} public static void Meto3(I3 i) {

System.out.println("I3");

i.metodo2();

} public static void Meto4(I4 i) {

System.out.println("I4");

i.metodo3();

}

public static void Metoafa(Alfa a) {

System.out.println("alfa");

a.metodoaf();

}

public static void main(String[] args) {

Bravo vio = new Bravo();

Meto1(vio);

Meto2(vio);

Meto3(vio);

Meto4(vio);

Metoafa(vio);

}

}

**Ex. 19**

interface Tossing {

void toss(int o);

}

interface GameType {

Tossing SpielErgeb();

}

class Coin implements Tossing {

Coin() { System.out.println(" Coin beschafft!"); }

public void toss(int i) {

if(i==1) System.out.println("H");

else if (i==0)

System.out.println("T");

}

}

class CoinFak implements GameType {

CoinFak() {System.out.println("CoinFak beschafft!"); }

public Tossing SpielErgeb() { return new Coin(); }

}

public class Main

{ public static Tossing spielen(GameType fabri) {

Tossing toso = fabri.SpielErgeb();

return toso;

}

public static void main(String[] args) {

Tossing A = spielen(new CoinFak());

A.toss(-3);

**Inner Classes Inner Classes Inner Classes**

**Ex. 1** class Aussen {

public Innen metodo() { return new Innen(); }

class Innen { public Innen() { System.out.println("Innen"); }

}

}

public class Main

{

public static void main(String[] args) {

Aussen uss = new Aussen();

Aussen.Innen ina = uss.metodo();

**Ex. 2** class Sequence {

private Object[] item;

private int next = 0;

private int i= 0;

public Sequence(int size) { item = new Object[size]; }

public void add(Object x) {

if (next < item.length)

item[next++] = x;

}

public Object Erhalten() { return item[i]; }

}

class Uma {

private String itu;

public Uma(String si) { itu = si; }

public String toString() {

return this.itu;

}

}

public class Main

{

public static void main(String[] args) {

Sequence Ina = new Sequence(5);

for(int i=0; i <5; i++){

Ina.add(new Uma("Eda"));

System.out.println(Ina.Erhalten());

}

}

}

**Ex. 3** class Aussen {

private String sui;

public Aussen(String u) { sui = u; }

public Innen metodo() { return new Innen(); }

class Innen { public Innen() { System.out.println("Innen"); }

public String toString() { return sui; }

}

}

public class Main

{

public static void main(String[] args) {

Aussen sulis= new Aussen("Friedrich");

Aussen.Innen ina = sulis.metodo();

System.out.println(ina);

**Ex. 4** class Sequencia {

public class Tina{

public Sequencia metoSeq() {

return Sequencia.this;

}

}

void Fernanda() { System.out.println("Fernanda"); }

public Tina metoTina() { return new Tina(); }

}

public static void main(String[] args) {

Sequencia senia = new Sequencia();

Sequencia.Tina seqtina = senia.metoTina();

seqtina.metoSeq().Fernanda(); \*/

**Ex. 5** class A {

class V { final int i = 800;

public V() { System.out.println("i = " + i); }

}

}

public static void main(String[] args) {

A aq = new A();

A.V vi = aq.new V();

**Ex. 6**

**Ex. 7** class G31{

private String sui;

private void metoG31() { System.out.println("metoG31"); }

class BV {

public String metoB(){

metoG31();

return sui = " # \*\_$&";

}

}

void metoGBV() {

BV beta = new BV();

beta.metoB();

System.out.println(sui);

}

}

public class Main

{

public static void main(String[] args) {

G31 Joao = new G31();

Joao.metoGBV();

G31.BV angus = Joao.new BV();

System.out.println(angus);

**Ex. 8 1st method (non-static)**

class G31{

class BV {

private final String gui= "Nihon" ;

}

public String meto() {

BV binal = new BV();

return binal.gui;

}

}

public class Main

{

public static void main(String[] args) {

G31 gigo= new G31();

System.out.println(gigo.meto());

**2nd Method (static variables )**

class G31{

class BV {

private final static String gui= "Nihon" ;

}

public static String meto() {

return BV.gui;

}

}

public class Main

{

public static void main(String[] args) {

System.out.println(G31.meto());

**Ex. 9** interface G31{

void meto();

}

class Adieu {

public G31 metodo1(String sui) {

class Rulis implements G31 {

public Rulis(String sui){}

public void meto() { System.out.println("Interface G31 meto"); }

}

return new Rulis(sui);

}

}

**Ex. 10** class Adieu {

public G31 metodo1(String sui) {

{ class Rulis implements G31 {

public Rulis(String sui){}

public void meto() { System.out.println("Interface G31 meto"); }

}

return new Rulis(sui);

}

}

} public static void main(String[] args) {

Adieu gigo= new Adieu();

G31 guan = gigo.metodo1("Ricardo");

guan.meto();

**Ex. 11** interface G31{

void meto();

}

class Adieu {

private class Rulis implements G31 {

public void meto() {System.out.println("Innen"); }

}

public G31 metodo1() { return new Rulis(); }

}

public class Main

{

public static void main(String[] args) {

Adieu gigo = new Adieu();

G31 gari = gigo.metodo1();

Adieu.Rulis f = gari; // **! Adieu.Rulis has private access in Adieu**

**Ex. 12**

abstract class BV { abstract public String metoB(); }

class G31{

private String sui;

public G31(String zan) { sui = zan; }

private void metoG31() { System.out.println("metoG31"); }

public BV Todo1() {

return new BV() {

public String metoB(){

metoG31();

return sui = " # \*\_$&";

}

};

}

void metoGBV() {

BV beta = Todo1();

beta.metoB();

System.out.println(sui);

}

}

public class Main

{ public static void main(String[] args) {

G31 Joao = new G31("Hanshu");

Joao.metoGBV();

**Ex. 13** interface G31 {

void meto();

}

class Adieu {

public G31 metodo1(String sui) {

return new G31() {

public void meto() {

System.out.println("Innen meto: " + sui);

}

};

}

}

public static void main(String[] args) {

Adieu ana = new Adieu();

G31 meng = ana.metodo1("Wert");

**Ex. 15**  class A1 {

private char seong;

private long jun;

A1() { System.out.println("Default Konst.");

}

A1(char cite, long hui) {

seong = cite;

jun = hui;

System.out.println(seong + " " + jun);

}

}

class Vita {

public A1 Metodo1 () {

return new A1() {};

}

public A1 Metodo1(char ui, long hui) {

return new A1( ui, hui) {};

}

}

public class Main

{ public static void main(String[] args) {

Vita v1 = new Vita();

A1 afa = v1.Metodo1();

A1 oma = v1.Metodo1('#',231);

**Ex. 16** interface Zyklus {

void Velo();

}

interface ZyklusFak {

Zyklus MetoZy();

}

class DreiZyklus implements Zyklus {

private DreiZyklus() {System.out.println("DreiZyklus beschafft!"); }

public void Velo() { System.out.println("Le velo courit!");

}

public static ZyklusFak fabrika = new ZyklusFak() {

public Zyklus MetoZy() { return new DreiZyklus(); }

};

}

public class Main

{ public static void ZyklusHerst(ZyklusFak fabrika) {

Zyklus z = fabrika.MetoZy();

}

public static void main(String[] args) {

ZyklusHerst(DreiZyklus.fabrika);

**Ex. 18, 19**

class A {

class B {

class J {

}

}

}

public class Main

{ private static class omega {

omega() { System.out.println("hoi"); }

protected static class renga {

renga() {System.out.println("Renga"); }

}

}

public static void main(String[] args) {

omega ega = new omega();

omega.renga sega = new omega.renga();

**Ex. 20** public class Main

{ interface Inter1 {

long hjuri();

class Test implements Inter1 {

public long hjuri() {

return -8;

}

}

}

public static void main(String[] args) {

System.out.println(new Inter1.Test().hjuri());

**Ex. 21**

interface Inter1 { **Non functional!**

static long hjuri(Inter1 in);

class Mango implements Inter1 {

public static long hjuri(Inter1 in) {

return -8;

}

class Test {

public static void Jullie(Inter1 in) {

hjuri(in);

}

}

}

}

**Ex. 22**

public class reverseSelector implements Selector {

private int i = items.length;

public boolean end() { return i==0; }

public Object current() { return items[i]; }

public void next() { if(i > 0) i--; }

}

**Ex. 23**

interface U {

void Todo();

void Todi();

void Toda();

}

class ACA implements U { public void Todo() { System.out.println("Todo Angeruft"); }

public void Todi() { System.out.println(this + "Todi"); }

public void Toda() { System.out.println("Todaa Angeruft"); }

public U MetoU() {

return new ACA() {};

}

class B {

U[] uro = { new ACA(), new ACA(), new ACA() };

public void Zufugung(U u, int oi) {

if(oi < uro.length) {

uro[oi] = u;

oi++;

}

}

public void Einsetzung(int i) {

uro[i] = null;

}

public void Trag() {

for( U u : uro) {

Todo();

Todi();

Toda();

}

}

}

}

public class Main

{ public static void main(String[] args) {

//I i1 = new I();

ACA a1 = new ACA();

ACA a2 = new ACA();

ACA.B b1 = a1.new B();

b1.Zufugung(a1, 0);

b1.Zufugung(a2, 1);

b1.Zufugung(a1, 2);

b1.Einsetzung(1);

System.out.println(b1.uro[0]);

System.out.println(b1.uro[1]);

System.out.println(b1.uro[2]);

b1.Trag();

**Ex. 24**

// In GreenhouseControls

private boolean Fan= false;

public class FanOn extends Event {

public FanOn(long delayTime) { super(delayTime); }

public void action() { //Hardware control code

Fan = true;

}

public String toString() {return "Fan is on"; }

}

//In GreenhouseController

// Event [] eventList = {...

gc.new FanOn(300)

**Ex. 26**

class Bakira {

class Pantro {

public Pantro(double hu){ hu = 0.082;

System.out.println(hu);

}

}

}

public class Main extends Bakira.Pantro

{ Main(Bakira bak) {

bak.super(3); // Calls Pantro constructor. Hence a double as argument

}

public static void main(String[] args) {

Bakira ara = new Bakira();

Main ii = new Main(ara);

**Containers -Collections-Iterators Containers -Collections-Iterators**

**Ex. 1**

import java.util.\*;

class Gebiou {

private int GebNo;

Gebiou(int oik) { GebNo = oik; }

public void hop(int GebNo) {

System.out.println(GebNo + " hopping!");

}

}

public class Main {

public static void main(String[] args) {

ArrayList geou = new ArrayList();

for (int i=0; i<3; i++)

geou.add(new Gebiou(i));

for (int i=0; i<3; i++)

((Gebiou)geou.get(i)).hop(i);

**Ex. 2**

public static void main(String[] args) {

Set<Integer> c = new HashSet<Integer>();

for(int i = 0; i < 10; i++)

c.add(i);

for(Integer i : c)

System.out.println(i + ", ");

**Ex. 4**

class CastGenerator {

static int i=0;

public String next() {

String[] san = {"Joao", "Juliano", "Sampaio" };

switch(i) {

default:

i= 0;

case 0:

i++;

return san[i-1];

case 1:

i++;

return san[i-1];

case 2:

i++;

return san[i-1];

}

}

}

public class Main {

static CastGenerator Asan = new CastGenerator();

static Collection Einfull(Collection<String> Kollekto) {

for(int i=0; i<3; i++)

Kollekto.add(Asan.next());

return Kollekto;

}

public static void main(String[] args) {

System.out.println(Einfull(new ArrayList<String>()));

System.out.println(Einfull(new LinkedList<String>()));

System.out.println(Einfull(new HashSet<String>()));

System.out.println(Einfull(new LinkedHashSet<String>()));

System.out.println(Einfull(new TreeSet<String>()));

**Ex. 7**

class Sonia {}

public class Main {

public static void main(String[] args) {

Sonia[] Artsel = { new Sonia(), new Sonia(),

new Sonia() };

List<Sonia> SoList = new ArrayList<Sonia>(Arrays.asList(Artsel));

System.out.println(SoList);

List<Sonia> subir = SoList.subList(0,2);

SoList.removeAll(subir);

System.out.println("14: " + SoList);

**Ex. 8**

public class Main {

public static void main(String[] args) {

List geou = new ArrayList();

for (int i=0; i<3; i++)

geou.add(new Gebiou(i));

Iterator<Gebiou> ita = geou.iterator();

int i= 0;

while(ita.hasNext() && i<3) {

Gebiou jeo = ita.next();

jeo.hop(i);

i++;

}

**Ex. 9 Non functional**

class Sequence { // p348

private Object[] items;

private int nxt = 0;

public Sequence(int size) { items = new Object[size]; }

public void add(Object x) {

if(nxt <items.length)

items[nxt++] = x;

}

}

public class Main {

public static void main(String[] args) {

Sequence Encia = new Sequence(10);

for (int i = 0; i < 10; i++)

Encia.add(Integer.toString(i));

Iterator<Sequence> ita = Encia.iterator(); // **Error. Cannot find symbol. Symbol: method iterator()**

while(ita.hasNext()) // iterator() is a method recognized by collections’ containers.

System.out.println(ita.next());

**Ex. 11**

private static int cou= 0;

Jong (String su) { sung = su;

cou++; }

public String toString() {

return sung;

}

}

public class Main {

static void Ausdrueck(Collection Ko) {

Iterator<Jong> ita = Ko.iterator();

while(ita.hasNext()) {

Jong Koli = ita.next();

System.out.println(Koli);

}

}

public static void main(String[] args) {

List<Jong> Sat1 = new ArrayList<Jong>(Arrays.asList(new Jong("Lao"), new Jong("Sung"),

new Jong("Zhe")));

Ausdrueck(Sat1);

**Ex. 12**

public class Main {

public static void main(String[] args) {

List<Integer> og1 = new ArrayList(Arrays.asList(1, 1, 3, -7, 5));

List<Integer> og2 = new ArrayList(Arrays.asList(2, 2, -49, -37, -21));

ListIterator<Integer> ita = og1.listIterator(5);

int i=0;

while(ita.hasPrevious()) {

// System.out.println(ita.previous());

og2.add(i,ita.previous());

i++;

}

System.out.println(og2);

**Ex. 14 Incomplete**

public static void main(String[] args) {

LinkedList<Integer> Lista = new LinkedList<Integer>();

// LinkedList<Integer> Sua = Integer.arrayList(4);// asa= { 6, 7, 97, 52};

List<Integer> ajax = new arrayList<Integer>();

ListIterator<Integer> ita = ajax.listIterator();

for (int k=0; k<5; k++) {

ajax.add(new Integer(k));

}

System.out.println(Lista);

**Ex. 15 Incomplete**

import java.util.\*;

class Staka<T> {

private LinkedList<T> storage = new LinkedList<T>();

public void push(T v) { storage.addFirst(v); }

public T peek() { return storage.getFirst(); }

public T pop() { return storage.removeFirst(); }

public boolean empty() { return storage.isEmpty(); }

public String toString() { return storage.toString(); }

}

public class Main

{

public static void main(String[] args) {

Staka<String> stani = new Staka<String>();

String sonia= "+U+n+c---+e+r+t---+a-+i-+n+t+y---+-+r+u--+l+e";

for(String s : sonia.split(" ")){

char d;

if ( s =="+")

stani.push(s);

else if (s =="-")

stani.pop();

} while(!stani.empty())

System.out.println(stani.pop() + " ");

System.out.println(sonia.split(" "));

**Ex. 16 Needs the mindview.net object Textfile**

public static void main(String[] args) {

Set<String> seio1 = new HashSet<String>();

Collections.addAll(seio1,"a e i o u y".split(" "));

System.out.println(seio1);

**Ex. 18, 24** Incomplete. Extract, sort, reinsert pairs to maps

public class Main

{

public static void main(String[] args) {

Map<String, Float> seo1 = new HashMap<String,Float>();

seo1.put("Gros1", new Float(98));

seo1.put("Gros2", new Float(-9));

seo1.put("Gros3", new Float(30));

System.out.println(seo1);

for(String s: seo1.keySet())

System.out.println(seo1.get(s)) ;

Set Song = new TreeSet();

Map LHM = new LinkedHashMap();

for(String sf:seo1.keySet()){

Song.add(seo1.get(sf));

}LHM.addAll(Song);

**Ex. 24** Incomplete

class Madamme { private String ui;

Madamme(String ios) { ui = ios; }

public String toString() { return this.ui; }

}

public class Main {

public static void main(String[] args) {

LinkedHashMap<String,Madamme> jioz = new LinkedHashMap();

jioz.put("Songhua ", new Madamme(" Soula"));

jioz.put("Jiaohua ", new Madamme(" Valentina"));

System.out.println(jioz);

**Ex. 28**

**Exceptions-Error Handling Exceptions-Error Handling**

**Ex. 1** class Ezzepsionato extends Exception {

public Ezzepsionato(String sun) {super(sun); }

}

public class Main

{ public void filenada() throws Ezzepsionato {

System.out.println("Innen filenada \n");

throw new Ezzepsionato("Divina");

}

public static void main(String[] args) {

Main Sia = new Main();

try {

Sia.filenada();

}

catch(Ezzepsionato enzo) {

enzo.printStackTrace(System.out);

//System.out.println("Innen catch! " + enzo );

}

finally {

System.out.println("Nach catch Satz");

}

}

}

**Ex. 2** class Testando {

public void Maria() { System.out.println("Innen Maria "); }

}

public class Main

{

public static void main(String[] args) {

Testando Tulaa = new Testando();

Tulaa = null;

try {

Tulaa.Maria();

}

catch (Throwable toro) {

toro.printStackTrace(System.out);

/\* Code for resumptionTestando tes = new Testando();

tes.Maria(); \*/

}

**Ex. 3** public static void main(String[] args) {

int[] A = { -2, 5, 3};

try {

for(int i =0; i<A.length; i++)

System.out.println(A[i]);

System.out.println(A[3]); // java.lang.ArrayIndexOutOfBoundsException: 3

// at Main.main(Main.java:10)

}

catch (Throwable toro) {

toro.printStackTrace(System.out);

}

**Ex. 4** class Essepsiona extends Exception {

String mon;

public Essepsiona(String sui) { mon = sui; }

public String toString() { return this.mon; }

}

public class Main

{

public static void main(String[] args) {

Essepsiona esi = new Essepsiona("Redondo! \n");

try {

System.out.println(esi);

throw esi;

}

catch (Essepsiona i) {

System.out.println("Innen catch");

}

**Ex. 7** import java.util.logging.\*;

import java.io.\*;

class Logging\_sonia extends Exception {

private static Logger logger =

Logger.getLogger("Logging\_sonia");

public Logging\_sonia() {

StringWriter trace = new StringWriter();

printStackTrace(new PrintWriter(trace));

logger.severe(trace.toString());

}

public String getMessage() {

return " Mesajo: " +" " + super.getMessage();

}

}

public class Main

{

public static void main(String[] args) {

try {

throw new Logging\_sonia();

}

catch (Logging\_sonia i) {

System.err.println("Innen catch " + i);

i.printStackTrace(System.out);

}

}

}

**Ex. 9** import java.util.logging.\*;

//import java.io.\*;

class Yolanda extends Exception {}

class Julietta extends Exception {}

class Divina {

public void filia() throws Yolanda, Julietta {

//throw new Julietta(); // **Error. Could not throw both exception types**

throw new Yolanda(); //**Only one type of the two exceptions could be thrown.**

}

}

public class Main

{

public static void main(String[] args) {

Divina Anna = new Divina();

try {

Anna.filia();

}

catch (Exception i) {

// System.err.println("Innen catch " + i);

i.printStackTrace(System.out);

**Ex. 10, 11**

class Ekzeption extends Exception {}

class Altro extends Exception {}

class IlMondo {

public void f() throws Altro {

try { g(); }

catch(Ekzeption ekz) {

throw new RuntimeException();

// throw new Altro();

}

}

private void g() throws Ekzeption {

System.out.println("@ \* &^");

if(true) throw new Ekzeption();

}

}

public class Main

{

public static void main(String[] args) {

IlMondo mo= new IlMondo();

try { mo.f(); }

catch(Altro ro) {

System.out.println("Altro gegriefen!");

}

**Ex. 13** public class Main

{

public static void main(String[] args) {

Divina Anna = new Divina();

try {

Anna.filia();

}

catch (Exception i) {

// System.err.println("Innen catch " + i);

i.printStackTrace(System.out);

} finally {

System.out.println("Final ausgefuehrt!");

}

**Ex. 14, 15** Same concept as OnOffSwitch.java and WithFinally.java

public class Main

{ static int j=-2;

public static void main(String[] args) {

try { j=-1;

System.out.println(j);

throw new RuntimeException();

} //finally {

j=0;

System.out.println(j);

// }

**Ex. 17**  Equivalent to Frog.java

class Alma {

Alma() { System.out.println("Alma beschafft");

}

static void depos() {

System.out.println(" Depose confirme");

}

}

public class Main {

Alma ag = new Alma();

public static void main(String[] args) throws Exception

try { if (true)

return;

Alma uri = new Alma();

System.out.println("Ende des try");

} finally {

Alma.depos();

}

**Ex. 18, 19**

public class Main

{ void f() throws VImExc {

throw new VImExc();

}

void depos() throws HoHEx {

throw new HoHEx();

}

public static void main(String[] args) {

try {

Main manos = new Main();

try { manos.f();

} catch(Exception exe) {

System.out.println(exe);

} finally {

manos.depos();

}

} catch(Exception exe) {

System.out.println(exe);

}

**Ex. 21**

class Bella extends Alma {

Bella() throws Kost {

try {

new Alma(); }

catch(Kost ko) {

System.out.println("gegriffen"); }

}

} // Prints: Exception in thread "main" Kost

         at Alma.<init>(Main.java:4)

         at Bella.<init>(Main.java:7)

         at Main.main(Main.java:22)

**Ex. 22**

class Kost extends Exception {}

class Alma {

Alma() throws Kost { throw new Kost(); }

}

public class Main {

public static void main(String[] args) throws Exception {

try {

Alma ag = new Alma();

}

catch(Kost ko) {

System.out.println("Truktor Ekzeption");

} finally {

System.out.println(" Finalisar");

}

**Ex. 23**

class Kost extends Exception {}

class DisoKlas {

public DisoKlas() { System.out.println("Diso cree!"); }

public void dispose() {

System.out.println("Obj depose!");

}

}

class Alma {

DisoKlas K1;

DisoKlas K2;

Alma() throws Kost {

try { K1= new DisoKlas();

try {

throw new Kost();

}

catch(Kost a){System.out.println("Kost wurde behandelt"); }

K2= new DisoKlas(); }

catch(Exception e){System.out.println(e); }

}

}

public class Main {

public static void main(String[] args) throws Exception {

Alma ag;

try {

ag = new Alma();

}

catch(Kost ko) {

System.out.println("Truktor Ekzeption");

**Ex. 24**

public class Main {

public static void main(String[] args) throws Exception {

try {

Alma ag = new Alma();

try { }

finally { ag.dispose(); }

//ag.K2.dispose();

} catch(Kost ko) {

System.out.println("Truktor Ekzeption");

}

**Ex. 27**

public static void main(String[] args) {

String[] siu = {"1nf", "fpa"};

try { System.out.println(siu[2]);

}

catch(Exception e) {

throw new RuntimeException();

}

**Ex. 28**

class Essepsiona extends RuntimeException {

String mon;

public Essepsiona(String sui) { mon = sui; }

public String toString() { return this.mon; }

}

public class Main

{

public static void main(String[] args) {

Essepsiona esi = new Essepsiona("Redondo! \n");

System.out.println(esi);

throw esi;

**Ex. 29**

class Essepsiona extends RuntimeException {

String mon;

public Essepsiona(String sui) { mon = sui; }

public String toString() { return this.mon; }

}

public class Main

{

public static void main(String[] args) {

Essepsiona esi = new Essepsiona("Redondo! \n");

System.out.println(esi);

throw esi;

**Ex. 30**

class Angst extends RuntimeException {}

class SZ extends Angst {}

class Wrappexce {

void throwRtexc(int id) {

try {

switch(id) {

case 0: throw new Angst();

case 1: throw new SZ();

default: return;

}

} catch(Exception e) {

throw new RuntimeException();

}

}

}

public class Main {

public static void main(String[] args) {

Wrappexce wce = new Wrappexce();

wce.throwRtexc(0);

/\*Or alternatively

try {

wce.throwRtexc(1);

}

catch(RuntimeException re) {

try {

throw re.getCause();

} catch(Angst a) {

System.out.println("Ang cat");

}

catch(Throwable bi) {

System.out.println("in cat");

}

}\*/

**Strings/Regular Expressions/regex classes & methods**

**Ex. 6**

class Tura {

private final int hui= 2;

// private float zhao= 2.0;

private final long hua= 87;

private double shang= 4.98;

public String toString(Tura Z){

return String.format("hui = %d, hua= %d, Shang = %f",

hui, hua, shang);

}

}

public class Main

{

public static void main(String[] args) {

Tura tressa = new Tura();

**Ex. 10, 11**

public static void main(String[] args) {

final String aero = "Java now has regular expressions ";

Matcher ma1 = Pattern.compile("n.w\\s +h(a|i)s").

matcher(aero);

final String kona = "Arline ate eight apples and one orange" +

" while Anita hadn't any";

Matcher ma1 = Pattern.compile("(?i)((^[aeiou])|(\ \s+[aeiou]))\ \w+?[aeiou] \\b").

matcher(kona);

while(ma1.find()) //n.w\\s+h(a|i)s

System.out.println(ma1.group() +" ");

**Ex. 13**

import java.util.regex.\*;

public class Main {

public static String Input="";

private static class Ausdruck {

private boolean regexgedrueckt = false;

private String regex;

Ausdruck(String regex) { this.regex = regex; }

void Druk\_Meto(String Bericht) {

if(!regexgedrueckt) {

System.out.println(regex);

regexgedrueckt = true;

}

System.out.println(Bericht);

}

}

static void pruefen(String s, String regex) {

Ausdruck aus= new Ausdruck(regex);

Pattern p = Pattern.compile(regex);

Matcher m = p.matcher(s;)

while(m.find())

aus.Druk\_Meto("find() '" + m.group() + "' start= "+ m.start() +

" end = " + m.end());

if(m.lookingAt())

aus.Druk\_Meto("lookingAt() start = "

+ m.start()) + "end = " + m.end());

if(m.matches())

aus.Druk\_Meto("matches() start = " + m.start() + " end = "

+ m.end());

}

public static void main(String[] args) {

for(String in : Input.split("\n")) {

System.out.println("input : " + in);

for(String regex : new String[] {"\\w\*ere\\w\*",

"\\w\*ever", "T\\w+", "Never.?!"})

pruefen(in, regex);

}

}

}

**Ex. 14**

public static void main(String[] args) {

String Eintrag =

"Diese!!unuebliche Nutzung!!Exklamations!!punkte";

System.out.println(Arrays.toString(Eintrag.split("!!")));

System.out.println(Arrays.toString(Eintrag.split("!!", 2)));

**P. 542 Test code of** TheReplacements.java

public static void main(String[] args) {

String s = " \*! Here's a block of text to use as input" +

" to the regular expression matcher. Note that we' ll" +

" first extract the block of text by looking for special delimiters"

+ " !\*/";

Matcher mInput =

Pattern.compile("\\\*!(.\*)!\\\*/", Pattern.DOTALL)

.matcher(s);

if(mInput.find())

s = mInput.group(1);

s = s.replaceAll(" {2}", " ");

s = s.replaceAll("(?m)^ +", "");

System.out.println(s);

s = s.replaceFirst("[aeiou]", "(VOWEL1))");

StringBuffer sbuf = new StringBuffer();

Pattern p = Pattern.compile("[aeiou]");

Matcher m = p.matcher(s);

while(m.find())

m.appendReplacement(sbuf, m.group().toUpperCase());

m.appendTail(sbuf);

System.out.println(sbuf);

**Ex. 20**

class Alfa {

private int i;

private long lo;

private double dod;

private float fo;

private String sini;

public Alfa(String sui) {

Scanner clsin = new Scanner(sui);

this.i = Integer.parseInt(clsin.nextLine());

// this.lo = Long.parseLong(clsin.nextLine());

this.dod = Double.parseDouble(clsin.nextLine());

this.fo = Float.parseFloat(clsin.nextLine());

this.sini = clsin.nextLine();

System.out.println(i+ "\t"+ lo + "\t" + dod +

"\n"+ fo +"\t" + sini);

}

public String toString() {

return i + "\t"+ lo + "\t" + dod +

"\n"+ fo +"\t" + sini;

}

}

public static void main(String[] args) {

Alfa ana = new Alfa("3\n 78\n 4.21\n Juan Ramon de la Venga");

System.out.println(ana);

**Type Information- RTTI Type Information- RTTI**

**P. 559**

import java.util.\*;

interface HasBatteries{}

class Toy {

Toy() { System.out.println("Zoi");}

Toy(int ur) { System.out.println("Zoi " +ur);}

}

class FancyToy extends Toy

implements HasBatteries {

FancyToy() { super(1); }

}

public class Uka {

public static void main(String []args){

Class c = null;

Toy sia = new Toy(-7);

/\*for(Class fi : c.getInterfaces())

System.out.println(fi);

try {

c = Class.forName("Typeinfo.toys.FancyToy");

} catch (ClassNotFoundException e) {

System.out.println("Nicht gefunden");

System.exit(1);

}

\*/

//Class up = c.getSuperclass();

Object o = null;

/\*try {

o = c.newInstance();

}

catch (InstantiationException e){

System.out.println("Keine Instanziation");

System.exit(1);

}

catch (IllegalAccessException su) {

System.out.println("Keiner Zugriff");

System.exit(1);

}\*/

}

}

**P. 567**

class Integ {

private static long counter;

private final long id = counter++;

public String toString() {

return Long.toString(id);

}

}

public class Main<T> {

private Class<T> type;

public Main(Class<T> type) { this.type = type; }

public List<T> Erg(int n) {

List<T> Re= new ArrayList<T>();

try {

for(int i = 0; i<n; i++ )

Re.add(type.newInstance());

}

catch (Exception e) {

throw new RuntimeException(e);

}

return Re;

}

public static void main(String[] args) {

Main<Integ> fl =

new Main<Integ>(Integ.class);

System.out.println(fl.Erg(12));

**Ex. 1, 2**

interface MexicoFC {}

interface Tequila {}

class nubira {

//nubira() {}

nubira(int i) {}

}

class almeyda extends nubira implements MexicoFC, Tequila{

almeyda() { super(2); }

}

public class Main {

static void printInfo(Class c) {

System.out.println(c.getName());

System.out.println(c.isInterface());

System.out.println(c.getSuperclass());

System.out.println(c.getCanonicalName());

}

public static void main(String[] args) {

Class c = null;

try {

c = Class.forName("almeyda");

}

catch(ClassNotFoundException e) {

System.out.println("Die Klasse wurde nicht gefunden!");

System.exit(1);

}

for(Class face : c.getInterfaces())

printInfo(face);

**Ex. 3**

abstract class Zeichnung {

void zeichnen() { System.out.println(this +

".zeichnen()"); }

//abstract public String toString();

}

class Zirkel extends Zeichnung {

//public String toString() { return "Zirkel"; }

}

class Rhomboid extends Zeichnung {

//public String toString() { return "Rhomboid"; }

}

public class Main {

public static void main(String[] args) {

Zeichnung z1 = new Rhomboid();

System.out.println(z1);

Class<Rhomboid> r0 = Rhomboid.class;

Zirkel r1 = (Zirkel)z1;

System.out.println(z1);

System.out.println(r1);

/\* Trying to downcast Rhomboid to Zirkel throws:

java.lang.ClassCastException: Rhomboid cannot be cast to Zirkel \*/

**Ex. 5, 6**

abstract class Zeichnung {

void zeichnen() { System.out.println(this + ".zeichnen()"); }

abstract public String toString();

abstract void rotate(Zeichnung z); /\*\*/

}

class Zirkel extends Zeichnung {

public String toString() { return "Zirkel"; }

void rotate(Zeichnung z) { }

}

class Rhomboid extends Zeichnung {

public String toString() { return "Rhomboid"; }

void rotate(Zeichnung z) { if(z.getClass().getName() == "Rhomboid")

System.out.println(z + " rotate");

} /\*\*/

}

public class Main {

public static void main(String[] args) {

Rhomboid r1 = new Rhomboid();

Zirkel z1 = new Zirkel();

r1.rotate(r1);

r1.rotate(z1);

**Ex. 7 Non functional! Problem to create at runtime the**

**appropriate object (using downcast from Object or Class classes?)**

class Candy {

public Candy() { System.out.println("Neues Ca"); }

// static { System.out.println("Candy geladen"); }

}

class Cookie {

//static { System.out.println("Cookie geladen!"); }

public Cookie() { System.out.println("Neues Co"); }

}

public class Main {

public static void main(String[] args) {

Pattern p = Pattern.compile(" ");

Scanner sc = new Scanner(System.in);

String[] s = p.split(sc.nextLine());

Class c = null;

//System.out.println(s[2]);

Object oje = null;

Class up = c.getClass();

try {

oje = up.newInstance();

}

catch(InstantiationException e) { System.exit(1); }

catch(IllegalAccessException e) { System.exit(1); }

if (s[2] =="Cookie"){

Cookie co= (Cookie)oje;

System.out.println(oje); }

else if(s[2] == "Candy"){

Candy ca= (Candy)oje;

System.out.println(oje);

}

**p.562-566**

public static void main(String[] args) {

Class<?> cam = Integer.class;

cam = float.class;

cam = Void.TYPE;

cam = Short.TYPE;

cam = Long.TYPE;

cam = boolean.class;

**Ex. 13**

import typeinfo.factory.\*;

public class Main {

public static void main(String[] args) {

TypeCounter counter = new TypeCounter(Part.class);

for (i<0; i<= 10; i++) {

part = Part.createRandom();

print(part.getClass().getSimpleName() + „ „);

counter.count(part);

}

print();

print(counter);

**Generics Generics Generics**

**Ex. 1, 2** class Annette {// Annette() { System.out.println("Annetta"); }

}

class Markela extends Annette {

private int i=0;

Markela () { System.out.println("Markela:" + ++i); }

}

class Holder3<lambda> {

private static int i = 0;

private lambda a;

public Holder3(lambda a) { this.a = a; }

public lambda get() { return a; }

public void set(lambda a) { this.a = a; }

/\* public void add(lambda a) { ++i;

this.a[i] = a; } \*/

}

public class Main {

public static void main(String[] args) {

Holder3<Annette> hong = new Holder3<Annette>(new Markela());

//hong.add(new Annette());

**Ex. 4**

interface Selector {

boolean end();

Object current();

void next();

}

class Sequence<Veda> {

private Veda[] entia;

private int nx = 0;

public Sequence(int so) { entia = (Veda[])new Object[so]; }

public void add(Veda x) {

if(nx < entia.length)

entia[nx++] = x;

}

private class SequenceSelector implements Selector {

private int i = 0;

public boolean end() { return i == entia.length; }

public Object current() { return entia[i]; }

public void next() { if(i < entia.length) i++; }

}

public Selector selector() {

return new SequenceSelector();

}

}

public class Main {

public static void main(String[] args) {

Sequence sequence = new Sequence(10);

for(int i = 0; i < 10; i++)

sequence.add(Integer.toString(i));

Selector selector = sequence.selector();

while(!selector.end()) {

System.out.print(selector.current() + " ");

selector.next();

}

}

}

**Ex. 5** class LinkedStack<T> {

private class Node {

T item;

Node next;

Node() { item = null; next = null; }

Node(T item, Node next) {

this.item = item;

this.next = next;

}

boolean end() { return item == null && next == null; }

}

private Node top = new Node(); // End sentinel

public void push(T item) {

top = new Node(item, top);

}

public T pop() {

T result = top.item;

if(!top.end())

top = top.next;

return result;

}

}

public class Main {

public static void main(String[] args) {

LinkedStack<String> lss = new LinkedStack<String>();

for(String s : "Phasers on stun!".split(" "))

lss.push(s);

String s;

while((s = lss.pop()) != null)

System.out.println(s);

}

}

**Ex. 6 ?**

class RandomList<T, akra, zi> {

private ArrayList<T> storage = new ArrayList<T>();

private Random rand = new Random(47);

public void add(T item) { storage.add(item); }

public T select() {

return storage.get(rand.nextInt(storage.size()));

}

}public class Main {

public static void main(String[] args) {

RandomList<Float, Integer, String> rs = new RandomList<Float, Integer, String>();

/\*\*/float[] k= {new Float(3.1), new Float(32.2)};

for(Float s: k)

rs.add(s);

for(int i = 0; i < 4; i++)

System.out.print(rs.select() + " ");

**Ex. 7**

class Ria {

private int count = 0;

public Integer next() { return fib(count++); }

private int fib(int n) {

if(n < 2) return 1;

return fib(n-2) + fib(n-1);

}

}

class Fibonacci {

Ria roula = new Ria();

}

public class Main {

public static void main(String[] args) {

Fibonacci gen = new Fibonacci();

for(int i= 0; i <16; i++)

System.out.println(gen.roula.next() + " ");

**Ex. 9, 10**

class Ria {

public <T, G, J> void f(T t, G g, J j) {

System.out.println(t.getClass().getName());

System.out.println(g.getClass().getName());

System.out.println(j.getClass().getName());

}

}

public class Main {

public static void main(String[] args) {

Ria Rania = new Ria();

Rania.f('q', Rania, -8.01);

**Ex. 11**

class Antoineta {

private static int i =10;

Antoineta() { i++; }

public String toString() { return "Annie" + i; }

}

class Fernanda {

private static int i =10;

Fernanda() { i++; }

public String toString() { return "Fenia" + i; }

}

class New {

public static <K,V> Map<K,V> map() {

return new HashMap<K,V>();

}

public static <T> List<T> list() {

return new ArrayList<T>();

}

public static <T> LinkedList<T> lList() {

return new LinkedList<T>();

}

public static <T> Set<T> set() {

return new HashSet<T>();

}

public static <T> Queue<T> queue() {

return new LinkedList<T>();

}

public void print() {}

} public class Main {

// Examples:

public static void main(String[] args) {

Map<Fernanda, List<Antoineta>> sls = New.map();

List<Antoineta> ls = New.list();

LinkedList<Fernanda> lls = New.lList();

Set<Fernanda> ss = New.set();

Queue<Antoineta> qs = New.queue();

**Ex. 12 Explicit type specification**

class New {

public static <K,V> Map<K,V> map() {

return new HashMap<K,V>();

}

static void Test(Map<Fernanda, Antoineta> Duetto){}

....

public static void main(String[] args) {

Map<Fernanda, List<Antoineta>> sls = New.map();

New.Test(New.<Fernanda, Antoineta>map());

**Ex. 14**

/\*Including the BasicGenerator file code and the Generator interface declaration: \*/

class Tonia { public String toString() { return " Ton"; }

}

public class Main {

public static void main(String[] args) {

Generator<Tonia> gen = new BasicGenerator<Tonia>(Tonia.class);

System.out.println(gen.next());

**Ex. 20 Generic method parameter type syntax error.**

interface Network {

void Julian();

void Marlene();

}

class Alfa implements Network {

public void Julian() { System.out.println("Julian"); }

public void Marlene() { System.out.println("Marlene"); }

public void Druck() { System.out.println("Alfa Methode!"); }

}

/\* \*/

class GeneKlass {

public static <? extends Alfa> void f(<? extends Alfa> argu) {

argu.Julian();

argu.Marlene();

argu.Druck();

}

}

public class Main {

public static void main(String[] args) {

// GeneKlass.f(new Alfa());

**Ex.21 ?**

class CaTyCa<T> {

Class<T> ki;

CaTyCa(Class<T> ki) {

this.ki = ki;

}

Map<String, Class<?>> Hata = new HashMap<String, Class<?>>();

public void addType(String typeName, Class<T>ki) {

Map<String, Class<T>> Hata = new HashMap<String, Class<T>>();

if(!Hata.containsKey(typeName))

Hata.put(typeName, typeName.class);

}

public <T> Class<T> crearNovo(String typeName) {

try {

}

catch (Exception e) {

throw RuntimeException(e);

}

}

public boolean f(Object sat) {

return ki.isInstance(sat);

}

}

public class Main {

**Ex.22 ?**

class Rica {

private char set;

private long sot;

Rica(char et, long ot) {

this.set = et;

this.sot = ot; // System.out.println("set: " + this.set + " sot: " + this.sot);

}

}

class Rebeka<Y> {

Class<Y> suo;

public Rebeka(Class<Y> sik) {

this.suo = sik;

}

}

public class Main {

<Y> Class<Y> Metodo(Class<Y> io) {

Class<Y> hik = Class.forName(i); //io.newInstance();

return hik;

}

public static void main(String[] args) {

Rebeka<Rica> R1 = new Rebeka<Rica>(Rica.class);

**Ex.23**

interface IAR<D> {

D crear(int u); }

class Fossa<D> {

private D x;

public <F extends IAR<D>> Fossa(F fi, int i) {

x = fi.crear(i); }

}

class Widg {

public static class Factory implements IAR<Widg> {

public Widg crear(int r) {

System.out.println("Integer r =" + r);

return new Widg();

}

}

}

public static void main(String[] args) {

new Fossa<Widg>(new Widg.Factory(),700);

**Ex. 25**

interface IA {}

interface IB {}

class Alfa<PO extends IA & IB> implements IA, IB {

PO po;

public Alfa(PO po) {

this.po = po;

}

}

public class Main {

static <PO extends IA> void Filo (Alfa<PO extends IA> aF) {

}

static <PO extends IB> void Bo (Alfa<PO extends BI> bo) {

}

public static void main(String[] args) {

Alfa a1 = new Alfa();

**Ex. 26**

public static void main(String[] args) {

Number[] nu = new Integer[4];

try {

nu[0] = new Number();

**Ex. 27**

class Fru {}

class Anana extends Fru {}

public class Main {

public static void main(String[] args) {

List<? extends Fru> fuli = new ArrayList<Anana>();

fuli.add(new Anana());

**Ex.28**

class Gene1<Ro> {

// Ro re;

// Gene1() { this.re = re; }

public void f1(Ro re) {

System.out.println(re);

}

}

class Gene2 <Ro> {

Ro ri;

public Ro f2() {

return ri;

}

}

public class Main {

static Gene1 r = new Gene1();

static <Ro> void fuGen1(Gene1<Ro> rosso) {

rosso.f1(r);

}

**Ex. 34 ? ?**

class Generico<RO extends Generico<RO>> {

RO rao;

Generico(RO ru) { rao = ru; }

abstract RO f1(RO ro);

public RO Me2() {

RO roMe2;

f1(RO ru){

roMe2 = ru ;

return roMe2;

}

return roMe2;

}

}

**Ex. 42**

interface UnaFun<RU,VO> { RU funcion(VO vik); }

class Anna {

private long hu= 7;

public long get() {

return hu;

}

public void set(long os) {

hu = os;

}

}

public class Main {

public static <RU,VO> List<RU> tras(Iterable<VO> ro, UnaFun<RU,VO> fio) {

List<RU> resu = new ArrayList<RU>();

for(VO v : ro)

resu.add(fio.funcion(v));

return resu; }

static class Ria implements UnaFun<Long, Anna> {

public Long funcion(Anna a) {

return a.get();

}

}

public static void main(String[] args) {

List<Anna> annie = new ArrayList<Anna>();

annie.add(new Anna());

annie.add(new Anna());

System.out.println(tras(annie, new Ria()));

**Arrays Arrays Arrays Arrays**

**Ex.3**

static double[][] funsi11(int n, double k, double i) {

double[][] dodo = new double[n][n];

double z = (i-k)/(n-1);

for(int zita = 0; zita < dodo.length; zita++) {

for (int u = 0; u < dodo.length; u++) {

dodo[zita][u] = k + u\*z;

}

}

return dodo;

}

static void funsi22(double[][] j) {

System.out.println(Arrays.deepToString(j));

}

public static void main(String []args) {

funsi22(funsi11(5, 3.5, 7.4));

**Ex.6**

static Metalo[][] funsi1(int to, int ko) {

Metalo[][] mi = new Metalo[to][ko];

for (int i = 0; i< to; i++)

for(int u= 0; u < ko; u++)

mi[i][u] = new Metalo();

return mi;

}

public static void main(String []args) {

System.out.println(Arrays.deepToString(funsi1(3, 2)));

**Enumerations p.1055 Code for test RoShamBo5 implementation**

public interface Competitor<T extends Competitor<T>> {

Outcome compete(T competitor);

}

enum Outcome { WIN, LOSE, DRAW }

public static <T extends Competitor<T>> void match(T a, T b) {

System.out.println(a + "vs. " + b + ": " + a.compete(b));

}

public static <T extends Enum<T> & Competitor<T>>

void play(Class<T> rsbClass, int size) {

for(int i = 0; i < size; i++)

match(Enums.random(rsbClass), Enums.random(rsbClass));

}

public enum RoShamBo5 implements Competitor<RoShamBo5> {

PAPER, SCISSORS, ROCK;

static EnumMap<RoShamBo5,EnumMap<RoShamBo5,Outcome>>

table = new EnumMap<RoShamBo5,

EnumMap<RoShamBo5,Outcome>>(RoShamBo5.class);

static {

for(RoShamBo5 it : RoShamBo5.values())

table.put(it,

new EnumMap<RoShamBo5,Outcome>(RoShamBo5.class));

initRow(PAPER, DRAW, LOSE, WIN);

initRow(SCISSORS, WIN, DRAW, LOSE);

initRow(ROCK, LOSE, WIN, DRAW);

}

static void initRow(RoShamBo5 it,

Outcome vPAPER, Outcome vSCISSORS, Outcome vROCK) {

EnumMap<RoShamBo5,Outcome> row =

RoShamBo5.table.get(it);

row.put(RoShamBo5.PAPER, vPAPER);

row.put(RoShamBo5.SCISSORS, vSCISSORS);

row.put(RoShamBo5.ROCK, vROCK);

}

public Outcome compete(RoShamBo5 it) {

return table.get(this).get(it);

}

public static void main(String[] args) {

RoShamBo.play(RoShamBo5.class, 20);

}

}