

Što je *boxing* (omatanje)?

a Postupak kojim se iz objekta omotača primitivnog tipa automatski dohvaća pohranjeni podatak.

b Postupak kojim se referenca određenog tipa pretvara u referencu na tip klase *Object*.

c Postupak pri kojem se na temelju podatka primitivnog tipa na hrdini stvara objekt koji pohranjuje kopiju tog podatka.

d Postupak pohrane objekata u Javine kolekcije.

Što je unboxing (odmotavanje)?

- a Postupak izvlačenja referenci na objekte koje su pohranjenjne u kolekcijama.
- b Postupak pri kojem se na temelju podatka primitivnog tipa na hripi stvara objekt koji pohranjuje kopiju tog podatka.
- c Postupak kojim se iz objekta omotača primitivnog tipa automatski dohvata pohranjeni podatak.
- d Postupak kojim se referenca određenog tipa pretvara u referencu na tip klase Object.

Score: 0.000 (=0.0%)

Id: 41095

Ako metoda ne može obraditi neku iznimku, a ne želimo baciti ru vrstu iznimke, moguće je:

a

omotati originalnu iznimku

b

u finally bloku uvijek je moguće obraditi iznimku

c

ništa od navedenog

d

nije moguće imati takvu iznimku

Koји од понуђених назива представља класе омотачких објеката?

a

Double

b

String

c

Int

d

Char

e

Float

U kojim linijama koda se događa *boxing* ?

```
int num = 10;           // 1
Integer base = num;     // 2
for (int i = 0; i < 3; i++, base++) {    // 3
    base = base + i;      // 4
}
```

a 3

b 1

c 2

d 4

U kojim linijama koda se događa unboxing ?

```
Integer i = new Integer(32);           // 1
if(i > 30) {                          // 2
    System.out.println("i > 30");      // 3
    i++;
}
```

a 3

b 2

c 1

d 4

Score: 0.500 (=100.0%)

Id: 41051

Koja je izjava točna, a odnosi se na niz naredbi koje su upisane u main metodu?

```
short s1 = 100;          // 1
Short s2 = s1;           // 2
Object o = s2;           // 3
Number n = o;            // 4
System.out.println(n);   // 5
```

a kod će se uspješno prevesti

b pogreška prevoditelja u liniji 3

c pogreška prevoditelja u liniji 1

d pogreška prevoditelja u liniji 2

e pogreška prevoditelja u liniji 4

Koja je izjava točna, a odnosi se na niz naredbi koje su upisane u main metodu?

```
double d1 = 10.5;           // 1
Double d2 = d1;             // 2
Object o1 = d2;             // 3
Integer i1 = (Integer) o1; // 4
System.out.println(i1);     // 5
```

a

pogreška prevoditelja u liniji 4

b

pogreška prevoditelja u liniji 1

c

kod će se uspješno prevesti ali će se dogoditi *ClassCastException* tijekom izvođenja

d

kod će se uspješno prevesti i nakon pokretanja ispisat će se 10

e

pogreška prevoditelja u liniji 2

Automatsko upravljanje resursima kod iznimki obavlja se posebnim oblikom try bloka koji se naziva:

a close on finally

b try-close-finally blok

c try with close

d try-close blok

e try with resources

Neka je razred R resurs u smislu kako to definira naredba try-with-resources. Razmatramo jednu konkretnu naredbu try-with-resources koja koristi takve resurse. Što je od slijedećega točno?

a Resursi se stvaraju u oblim zgradama nakon što se u vitičastim zgradama napisanim nakon ključne riječi try definira programski kod

b Ne trebamo pisati programski kod u bloku finally koji brine o zatvaranju resursa

c Resursi se stvaraju u vitičastim zgradama odmah nakon ključne riječi try

d Kada koristimo resurse, ne smijemo pisati blok finally

e Reference na korištene resurse dostupne su nam u blokovima catch i finally

imamo sljedeći kod:

```
public class A extends RuntimeException {}  
public class B extends A {}  
public class C extends A {}
```

Metoda m može baciti iznimke B i C. Koji kod je ispravan ako želimo obraditi iznimke B i C s istim kodom?

a

```
try {  
    m();  
} catch (B | C e) {  
    // obrada  
}
```

b

```
try {  
    m();  
} catch (A a) {  
    // obrada  
}
```

c

```
try {  
    m();  
} catch (A a) {  
    // obrada 1  
} catch (B b) {  
    // obrada 2  
} catch (C c) {  
    // obrada 3  
}
```

d

```
try {  
    m();  
} catch (C c) {  
    // obrada 1  
} catch (B b) {  
    // obrada 2  
} catch (A a) {  
    // obrada 3  
}
```

e

```
try {  
    m();  
} catch (RuntimeException re) {  
    // obrada  
}
```

Imamo sljedeći kod:

```
public class A extends RuntimeException {}  
public class B extends A {}  
public class C extends A {}
```

Metoda m može baciti iznimke B i C. Koji kod je ispravan ako želimo obraditi iznimke B i C različitim kodom?

a

```
try {  
    m();  
} catch (C c) {  
    // obrada 1  
} catch (B b) {  
    // obrada 2  
} catch (A a) {  
    // obrada 3  
}
```

Imamo sljedeći kod:

```
public class A extends RuntimeException {}  
public class B extends A {}  
public class C extends A {}
```

Metoda m može baciti iznimke B i C. Koji kod je ispravan ako želimo obraditi iznimku A posebno od iznimki B i C?

a

```
try {  
    m();  
} catch (RuntimeException re) {  
    // obrada  
}
```

b

```
try {  
    m();  
} catch (A a) {  
    // obrada 1  
} catch (B b) {  
    // obrada 2  
} catch (C c) {  
    // obrada 3  
}
```

c

```
try {  
    m();  
} catch (B | C e) {  
    // obrada  
}
```

d

```
try {  
    m();  
} catch (C c) {  
    // obrada 1  
} catch (B b) {  
    // obrada 2  
} catch (A a) {  
    // obrada 3  
}
```

e

```
try {  
    m();  
} catch (A a) {  
    // obrada  
}
```

```
public class Point<T> {  
    ...  
    public <V> void printWith(Point<V> another) {  
        // što je T, a što je V u ovom trenutku?  
        System.out.format("first: %s second %s %n", this.toString(), another.toString());  
    }  
}
```

Označite točne odgovore vezane za pitanje što je T, a što je V u ovom trenutku? Ako metodu `printWith` pozovemo iz sljedećeg programskog odsječka:

```
Point<Integer> x = new Point<>(2, 3);  
Point<String> y = new Point<>("A", "B");  
x.printWith(y); // što je T, a što V prilikom ovog poziva metode?
```

a T je isto što i V

b V je String

c T je Integer

d V je Point<String>

e T je Point<Integer>

```
System.out.format("first: %s second %s \n", this.toString(), another.toString());
```

```
}
```

```
}
```

Označite točne odgovore vezane za pitanje što je T, a što je V u ovom trenutku? Ako metodu `printWith` pozovemo iz sljedećeg programskog odsječka:

```
Point<Integer> x = new Point<>(2, 3);
Point<String> y = new Point<>("A", "B");
x.printWith(y); // što je T, a što V prilikom ovog poziva metode?
```

a

T je Point<Integer>

b

V je Point<String>

c

V je String

d

T je Integer

e

T je isto što i V

Score: 0.000 (-0.0%)

Id: 41054

Klasa IntegerPoint je definirana kako je navedeno. Pomoću kojih naredbi možemo kreirati objekt tipa IntegerPoint?

```
public class IntegerPoint {  
    private Integer x, y;  
    public IntegerPoint(Integer x, Integer y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

a

```
IntegerPoint ip = new IntegerPoint(new Integer(2), new Integer(3));
```

b

```
IntegerPoint ip = new IntegerPoint(new Short((short) 2), new Short((short) 3));
```

c

```
IntegerPoint ip = new IntegerPoint("2", "3");
```

d

```
IntegerPoint ip = new IntegerPoint(Integer.valueOf(2), Integer.valueOf(3));
```

e

```
IntegerPoint ip = new IntegerPoint(Integer.parseInt("2"), Integer.parseInt("3"));
```

Razmotrite sljedeći kod koji definira klasu Spremnik. Iznimke ArithmeticException i IllegalArgumentException izvedene su iz iznimke RuntimeException.

```
public class Spremnik implements AutoCloseable {
    private String ime;
    public Spremnik(String ime) {
        if(ime.length()==0)
            throw new IllegalArgumentException();
        this.ime = ime;
        System.out.print("Stvoren " + ime + ";");
    }
    public void close() {
        System.out.print("Zatvoren " + ime + ";");
    }
    public void write(int i) {
        if(i<0) throw new RuntimeException();
        if(i==0) throw new ArithmeticException();
    }
}
```

Ako je u metodi main neke klase napisan sljedeći kod, što će biti ispisano kao rezultat njegova izvođenja?

```
try {
    try(Spremnik s1 = new Spremnik("pero"); Spremnik s2 = new Spremnik("ana")) {
        System.out.print("N1;");
        s1.write(5);
        System.out.print("N2;");
        s2.write(-1);
        System.out.print("N3;");
    }
    catch(ArithmeticException ex) {
        System.out.print("AE;");
    }
    finally {
        System.out.print("FI1;");
    }
}
catch(RuntimeException ex) {
    System.out.print("RE;");
}
```

```
catch(RuntimeException ex) {  
    System.out.print("RE;");  
}  
finally {  
    System.out.print("FI2;");  
}
```

- a** Stvoren pero;Stvoren ana;N1;N2;Zatvoren ana;Zatvoren pero;AE;FI1;FI2;
- b** Stvoren pero;Stvoren ana;N1;N2;Zatvoren ana;Zatvoren pero;FI1;RE;FI2;
- c** Stvoren pero;Stvoren ana;N1;N2;N3;FI1;FI2;
- d** Ništa. Program se neće prevesti jer nakon oble zagrade iza ključne riječi try ne možemo stvarati primjerak klase Spremnik.
- e** Stvoren pero;Stvoren ana;N1;N2;Zatvoren ana;Zatvoren pero;AE;FI1;RE;FI2;
- f** Stvoren pero;Stvoren ana;N1;Zatvoren ana;Zatvoren pero;AE;FI1;FI2;
- g** Stvoren pero;Stvoren ana;N1;N2;N3;Zatvoren ana;Zatvoren pero;FI1;FI2;
- h** Stvoren pero;Stvoren ana;N1;N2;AE;FI1;FI2;

a

Stvoren pero;Stvoren ;N1;N2;N3;Zatvoren ;Zatvoren pero;FI1;FI2;

b

Stvoren pero;Stvoren ;N1;N2;Zatvoren ;Zatvoren pero;AE;FI1;RE;FI2;

c

Stvoren pero;Zatvoren pero;FI1;RE;FI2;

d

Ništa. Program se neće prevesti jer nakon oble zagrada iza ključne riječi try ne možemo stvarati primjerak klase Spremnik.

e

Stvoren pero;Stvoren ;N1;N2;N3;FI1;FI2;

f

Stvoren pero;Stvoren ;N1;N2;AE;FI1;FI2;

g

Stvoren pero;Stvoren ;N1;N2;Zatvoren ;Zatvoren pero;FI1;RE;FI2;

h

Stvoren pero;Stvoren ;N1;Zatvoren ;Zatvoren pero;AE;FI1;FI2;

Ako je klasa Gen parametrizirana po tipu T, tj. početak definicije klase glasi class Gen<T>, te postoji klasa Mapa parametrizirana po 2 tipa (npr. class Mapa<T1, T2>) što od navedenog je moguće definirati unutar klase Gen?

a <V> Mapa<T, V> map;

b Gen<T> g;

c T obj1;

d <V> void m7(T t, V v) { }

e static <U, V> void mv(Mapa<U, V> map) { }

Score: 0.000 (=0.0%)

Id: 41074

Automatsko upravljanje resursima može se koristiti s klasama koje implementiraju neko od sljedećih sučelja:

a Resource

b Finally

c AutoCloseable

d Throwable

e Closeable

Razred Main želimo parametrizirati po tipu koji je izведен iz apstrakne klase Number i implementira sučelje MojeSucelje. Odaberite ispravno rješenje

a

```
class Main<T extends Number & MojeSucelje> {}
```

b

```
class Main<T extends Number> implements MojeSucelje {}
```

c

```
class Main<T extends Number implements MojeSucelje> {}
```

d

```
class Main<T> extends Number implements MojeSucelje {}
```

e

```
class Main<T & Number & MojeSucelje> {}
```

Iznimke koje su izvedene iz klase *RuntimeException* pripadaju u porodicu

- a** provjeravane iznimke
- b** sintaksne iznimke
- c** grube iznimke
- d** neprovjeravane iznimke
- e** iznimke povezivanja

Score: 0.500 (=100.0%)

Id: 41108

Neprovjeravane iznimke (engl. unchecked exceptions) izvedene su iz klase

a

OutOfMemoryError

b

LinkageError

c

AssertionError

d

Error

e

ClassFormatException

f

RuntimeException

Koji od navedenih tipova iznimki spadaju u neprovjeravane iznimke?

a NumberFormatException

b IOException

c SQLException

d NullPointerException

e IndexOutOfBoundsException

Neka je razred R resurs u smislu kako to definira naredba try-with-resources. Razmotrite sljedeći kod napisan u metodi main.

```
try (R r = new R()) {  
}
```

Što je od sljedećega točno?

- a** kod se neće prevesti jer ne smije imati blok finally
- b** kod se neće prevesti jer nedostaje blok catch
- c** kod se neće prevesti jer nismo zatvorili resurs
- d** try-konstrukt je ispravan i moći će se prevesti

Neka je razred R resurs u smislu kako to definira naredba try-with-resources. Razmotrite sljedeći kod napisan u metodi main.

```
try (R r = new R()) {  
}  
catch(IOException ex) {}
```

Što je od sljedećega točno?

a

try-konstrukt je ispravan i moći će se prevesti

b

kod se neće prevesti jer ne smije imati blok catch

c

kod se neće prevesti jer nismo zatvorili resurs

d

kod se neće prevesti jer nedostaje blok finally

Ako u programskom odsječku nalik donjoj skici metoda M1 baca iznimku tipa *RuntimeException* koje od naredbi će se izvršiti

```
try{
    try{
        M1();
    }
    catch (NullPointerException exc){
        System.out.println("CATCH1");
    }
    finally {
        System.out.println("F1");
    }
    System.out.println("T1");
}
catch (IllegalArgumentException exc){
    System.out.println("CATCH2");
}
finally {
    System.out.println("F2");
}
System.out.println("T2");
```

a T1

b CATCH1

c T2

d CATCH2

e F1

f F2

Ako u programskom odsječku nalik donjoj skici metoda M1 baca iznimku tipa *NullPointerException* koje od naredbi će se izvršiti

```
try{  
    try{  
        M1();  
    }  
    catch (NullPointerException exc){  
        System.out.println("CATCH1");  
    }  
    finally {  
        System.out.println("F1");  
    }  
    System.out.println("T1");  
}  
catch (NullPointerException exc){  
    System.out.println("CATCH2");  
}  
finally {  
    System.out.println("F2");  
}  
  
System.out.println("T2");
```

a T1

b CATCH1

c F2

d CATCH2

e T2

f F1

Ako u programskom odsječku nalik donjoj skici metoda M1 baca iznimku tipa *IllegalArgumentException* koje od naredbi će se izvršiti

```
try{  
    M1();  
    System.out.println("T1");  
}catch (Exception exc){  
    System.out.println("CATCH");  
}  
finally {  
    System.out.println("F");  
}  
System.out.println("T2");
```

a T1

b T2

c CATCH

d F

Što sve od navedenog vrijedi za sljedeću metodu

```
public static <T extends Comparable<T>> int metoda(T[] polje, T element) {  
    //neki kod  
}
```



- a** S obzirom na to da metoda vraća cijelobrojnu vrijednost, argumenti *polje* i *element* moraju biti parametrizirani kao numerički tipovi
- b** Parametrizirana je po tipu koji implementira sučelje Comparable
- c** Ako se za *polje* ili *element* pošalje null, program će izbaciti iznimku NullPointerException.
- d** U kodu metode je moguće napisati naredbu `int x = element.compareTo(element);`
- e** Ništa od navedenog
- f** U kodu metode je moguće napisati naredbu `int x = polje[neki_valjani_indeks].compareTo(element);`

Iznimka *NumberFormatException* pojavljuje se u situaciji:

- a** ništa od navedenog
- b** kada ispis broja na standardni izlaz nije u skladu s konvencijama iz specifikacije Java
- c** kada aplikacija pokuša na osnovi sadržaja stringa stvoriti jedan od numeričkih tipova, ali string nije odgovarajućeg formata
- d** kada se dogodi neispravna aritmetička operacija numeričkih vrijednosti (zbrajanje, oduzimanje, množenje, zrajanje)
- e** kada se pokušaju zbrojiti dva broja, ali njihov zbroj premašuje maksimalnu vrijednost (tj. Integer.MAX_VALUE)

Neka je razred R resurs u smislu kako to definira naredbu try-with-resources. Razmatramo jednu konkretnu naredbu try-with-resources koja koristi takve resurse. Što je od sljedećega točno?

- a** Resursi se stvaraju u vitičastim zagradama odmah nakon ključne riječi try
- b** Kada koristimo resurse, ne smijemo pisati blok finally
- c** Resursi se stvaraju u oblim zagradama nakon što se u vitičastim zagradama napisanim nakon ključne riječi try definira programski kod
- d** Ako naredba treba stvoriti više resursa i jedan od njih izazove iznimku, prethodno stvorenii resursi bit će zatvoreni
- e** Reference na korištene resurse dostupne su nam u blokovima catch i finally

Označite točne tvrdnje

a

Klasa *Exception* i njen podstavak modelira opise iznimnih situacija od kojih se očekuje da je moguć oporavak.

b

Klasa *Exception* i njen podstavak modelira opise iznimnih situacija od kojih se **ne** očekuje da je moguć oporavak.

c

Klasa *Error* i njen podstavak modelira opise iznimnih situacija od kojih se očekuje da je moguć oporavak.

d

Klasa *Error* i njen podstavak modelira opise iznimnih situacija od kojih se **ne** očekuje da je moguć oporavak.

Što od navedenog vrijedi za kod napisan na sljedeći način:

```
public class Klasa<T> {  
    public static void metoda(Klasa<?> nekiObjekt) {  
        System.out.println(nekiObjekt.toString());  
    }  
}
```

- a** Kod je ispravan
- b** Kod je neispravan, jer nedostaje <?> između static i void
- c** Kod je neispravan, zašto što nije moguć poziv statičke metode ako je klasa parametrizirana.
- d** Kod je neispravan, jer je klasa parametrizirana s <T> pa kasnije korištenje <?> nije dozvoljeno.
- e** Ništa od navedenog
- f** Kod je sintaksno neispravan, jer se upitnik ne može koristiti za parametrizaciju

```
public <V> void metoda(V nekiObjekt) {  
    System.out.println(nekiObjekt.toString());  
}
```

Za navedenu parametriziranu metodu vrijedi sljedeće:

a Sintaksno nije ispravna jer je <V> višak

b Postoji klasa V koja nadjačava metodu *toString()*, tj.

```
class V {  
    ...  
    @Override  
    public String toString() { ... }  
}
```

c Zahvaljujući <V> metoda se može pozvati staticki nad klasom

d Metoda nije ispravna jer *nekiObjekt* ne mora nužno imati metodu *toString()*

e Uzrokovat će iznimku tipa *NullPointerException* ako je *nekiObjekt* null

```
float sum = 0;
for(int i=3; i>=0; i--){
    try{
        System.out.println(i);
        sum+= 1/i;
    }
    catch(NumberFormatException, ArrayIndexOutOfBoundsException exc) {
        System.out.println("NumberFormatException or ArrayIndexOutOfBoundsException");
    }
    catch(ArithmetricException exc) {
        System.out.println("ArithmetricException");
    }
}
System.out.println("Done");
```

a

321NumberFormatException or ArrayIndexOutOfBoundsExceptionDone

b

3210ArithmetricExceptionDone

c

321ExceptionDone

d

Program se neće izvesti zbog sintaksne pogreške, iznimke se ne ulančavaju sa zarezom nego sa znakom |.

321ArithmetricExceptionDone

Ako je klasa Gen parametrizirana po tipu T (tj. `class Gen<T>`) te je zadana metoda T m(T t), koje od sljedećih naredbi su moguće u metodi m?

a

```
return (T[]) new Object[10];
```

b

```
System.out.println(t.toString());
```

c

```
t = (T[]) new Object[10];
```

d

```
return t;
```

e

```
return new T();
```

Ako je klasa Gen parametrizirana po tipu T, (tj. početak definicije klase glasi `class Gen<T>`) te postoji parametrizirana klasa *MyList* što od navedenog je moguće definirati unutar klase Gen

a `<V> void m4(T t, Gen<Gen<V>> v) { }`

b `static MyList<T> list;`

c `static <V> void m(MyList<V> list) { }`

d `MyList<MyList<T>> listOfList;`

e `T obj1;`

Ako je klasa Gen parametrizirana po tipu T, (tj. početak definicije klase glasi `class Gen<T>`) što od navedenog je moguće definirati unutar klase Gen?

Napomena: Klasa List je također parametrizirana po nekom tipu, a klasa Map je parametrizirana po 2 tipa.

a

`Gen<Gen<Gen<Gen<Integer>>>> ggggi;`

b

`List<Gen<int>> lgi;`

c

`Gen<List<Gen<List<Integer>>>> glglg;`

d

`Gen<list<int>> gli;`

e

`Map<Gen<Integer>, Gen<Double>> map;`

Ako je klasa Gen parametrisirana po tipu T, (tj. početak definicije klase glasi `class Gen<T>`) što od navedenog je moguće definisati unutar klase Gen?

Napomena: Sučelje List je također parametrisirana po nekom tipu.

a

`List<Gen<Integer>> lgi;`

b

`Integer<Gen> ig;`

c

`Gen<double> gd;`

d

`Gen<List<Integer>> gli;`

e

`Gen<Integer> gi;`

Što sve od navedenog vrijedi za provjeravane iznimke?

a

Modeliraju situacije za koje želimo da se eksplicitno obrađuju

b

Takve iznimke nisu fatalne za aplikaciju (oporavak je moguć)

c

Modeliraju situacije koje se mogu javiti na svakom koraku izvođenja programa

d

Nije nužno ugraditi upravljanje takvim iznimkama

e

Mora ih se eksplicitno obradivati ili naglasiti da se prosljeđuju dalje