Laboratorium 7

CEL: Kontenery. Typy sparametryzowane.

Zadanie#1. Przeanalizuj poniższy kod.

```
"name:john;;sex:m;;".split(";");
```

Co zwróci powyższy fragment kodu?

a)	Tablicę typu `String` zawierająca 3 elementy
b)	Listę zawierająca 2 elementy
c)	Listę zawierająca 4 elementy
d)	Tablicę typu `String` zawierająca 4 elementy
e)	Tablicę typu `String` zawierająca 2 elementy

■

Zadanie#2. Klasa `Hashtable` implementuje interfejs kolekcji (jedna odpowiedź):

a)	`Table`
b)	`List`
c)	`Set`
d)	`SortedSet`
e)	`Map`

■

Zadanie#3. Czy `Set` jest kolekcją, która nie umożliwia umieszczania duplikujących się elementów?

a)	Tak
b)	Nie

■

Zadanie#4. Który z interfejsów kolekcji umożliwia przechowywanie elementów ułożonych sekwencyjnie (jedna odpowiedź)?

a)	`java.util.List`
b)	`java.util.Map`
c)	`java.util.Collection`
d)	`java.util.Set`

■

Zadanie#5. Chcesz przechowywać niewielką ilość danych i uzyskiwać do nich szybko dostęp. Nie potrzebujesz sortować tych danych, unikalność nie jest istotna, a dane pozostaną raczej niezmienne. Wskaż, jaka struktura danych może być najbardziej odpowiednia dla tego wymagania (jedna odpowiedź)?

a)	`TreeSet`
b)	`HashMap`
c)	`LinkedList`
d)	`Array`

■

Zadanie#6. Przeanalizuj poniższy kod.

```
01: import java.util.*;
02:
03: public class Test {
04:     public static void main(String[] args) {
05:         TreeSet map = new TreeSet();
06:         map.add("one");
07:         map.add("two");
08:         map.add("three");
09:         map.add("four");
10:         map.add("one");
11:         Iterator it = map.iterator();
12:         while (it.hasNext()) {
13:             System.out.print(it.next() + " ");
14:         }
15:     }
16: }
```

Wskaż, jaki wynik zostanie wyświetlony na konsoli?

a)	Błąd kompilacji
b)	`one two three four`
c)	`four three two one`
d)	`one two three four one`
e)	`one four three two one`
f)	`four one three two`
g)	Wyjątek podczas uruchomienia
h)	Porządek wyświetlania jest nieprzewidywalny

■

Zadanie#7. Przeanalizuj poniższy fragment kodu.

```
List<Integer> list = Collections.emptyList();
list.add(1);
list.add(2);
System.out.println(list.size());
```

Jaki będzie wynik wykonania?

a)	`0`
b)	`2`
c)	Wyjątek podczas uruchomienia
d)	Błąd kompilacji

■

Zadanie#8. Przeanalizuj poniższy kod.

```
01: public class Test {
02:   public static void main(String[] args) {
03:     Set s = new TreeSet();
04:
05:     s.add(new Integer(1));
06:     s.add("2");
07:     s.add(new Integer(3));
08:
09:     for(Iterator theIterator=s.iterator(); theIterator.hasNext(); ) {
10:             System.out.print(theIterator.next());
11:         }
12:   }
13: }
```

Wskaż, jaki wynik zostanie wyświetlony na konsoli?

a)	`123`
b)	Porządek wyświetlania jest nieprzewidywalny
c)	Wyjątek podczas uruchomienia
d)	Błąd kompilacji

■

Zadanie#9. Które z poniższych stwierdzeń jest/są prawdziwe?

a)	`Hashtable` jest podklasą `Dictionary`
b)	`ArrayList` jest podklasą `Vector`
c)	`LinkedList` jest podklasą `ArrayList`
d)	`Stack` jest podklasą `Vector`

■

Zadanie#10. Napisz nazwę klasy będącej najbardziej wydajną implementacją kontenera `Set` dla typów wyliczeniowych (`enum`).

■

Zadanie#11. Napisz własną implementację komparatora dla kontenera `TreeMap<String,String>` sortującego w porządku rosnącym kluczy `K` od ostatniego znaku klucza. Przedstaw kod całościowo.

■

Zadanie#12. Napisz kod klasy implementujący (`implements`) poniższy interfejs `IMinMax` **z przekazaniem parametru typu** (*sic!*). Przedstaw kod całościowo.

```
interface IMinMax< T extends Number > {
     T min( T[ ] tab ); //min. wartość w tablicy
     T max( T[ ] tab ); //max. wartość w tablicy
}
```

■

Zadanie#13. Przeanalizuj poniższy kod.

```
class NonGen {
     Object object;
     NonGen( Object o ) {
          object = o;
     }
     Object getObject( ) {
          return object;
     }
     void showType( ) {
          System.out.println(object.getClass().getName());
     }
}
```

Zastąp kod klasy `NonGen` nową klasą sparametryzowaną `Gen<T>`, gdzie `T` zastępuje `Object`. Przedstaw kod całościowo.

■

Zadanie#14. Przeanalizuj poniższy fragment kodu.

```
public class Example {
     ...
}
```

W klasie `Example` utwórz konstruktor o dwóch różnie parametryzowanych argumentach wywołania (bez parametryzowania klasy `Example`). Przedstaw kod całościowo.

■

Zadanie#15. Przeanalizuj poniższy kod.

```
class GenericsDemo<T>{
     private static T demo;

     public void myApplicationDemo( ) {
         System.out.println( "Executing My Application" );
     }
}
```

Czy powyższy kod zostanie skompilowany poprawnie?

a)	Tak
b)	Nie

■

Zadanie#16. Który typ `Map` wykorzystuje synchronizację dostępu?

a)	`HashMap<K,V>`
b)	`LinkedHashMap<K,V>`
c)	`Hashtable<K,V>`
d)	`Set<K,V>`

■

Zadanie#17. Przeanalizuj poniższy kod.

```
import java.util.*;

public class InstanceTester{
   public static  <E> void tester(List<E> list) { //Line 1
      if (list instanceof ArrayList<Integer>) {   //Line 2
         System.out.println("Test Successsful");
      }
   }
   public static void main(String... args){
      List<String> str = new ArrayList< >(); //Line 3
      InstanceTester.tester(str);
   }
}
```

Jaki będzie wynik wykonania?

a)	Błąd kompilacji w `//Line 1`
b)	Błąd kompilacji w `//Line 2`
c)	Błąd kompilacji w `//Line 3`
d)	Zostanie wyświetlone `"Test Successsful"`

■

Zadanie#18. Jaka jest główna różnica pomiędzy `StringBuffer` a `StringBuilder`, która wpływa na wydajność kodu? Wyjaśnij szczegółowo różnicę i jej wpływ na wydajność.

■

Poprawnych odpowiedzi	*Ocena*
<0>	n/k
<1, 10>	2.0
<11, 12>	3.0
<13, 14>	3.5
<15, 16>	4.0
<17>	4.5
<18>	5.0