Imię i nazwisko .....

Egzamin - Programowanie Obiektowe
II rok informatyki, studia pierwszego stopnia, niestacjonarne
Termin zerowy – 21.01.2017

Instrukcja: test składa się z 18 pytań jednokrotnego wyboru. Za poprawną odpowiedź: plus 3 pkt. Za błędną odpowiedź: minus 2 pkt. Brak odpowiedzi: 0 pkt. Punktacja: bardzo dobry (5,0) – 91-100%; dobry plus (4,5) – 80-90%; dobry (4,0) – 70-79%; dostateczny plus (3,5) – 60-69%; dostateczny (3,0) – 40-59%; niedostateczny (2,0) – 0-39%.

---

1. Jeśli `Osoba` i `Student` są klasami, która z konstrukcji nie jest dopuszczalna?

A `class Dwudziestolatek : Osoba, Student`
B `class Dwudziestolatek : Osoba`
C `class Dwudziestolatek : Osoba, ICloneable`
D `class Dwudziestolatek : Student, ICloneable`

---

2. Która z poniższych instrukcji nie będzie stanowiła konstruktora w klasie `Samochod`?

A `public Samochod() { }`
B `public void Samochod() { }`
C `public Samochod(string marka) { }`
D `public Samochod() { this.marka = "Ford"; }`

---

3. Jaki modyfikator dostępu powinniśmy użyć do pola/metody, aby było ono dostępne dla klasy potomnej?

A `private`
B `protected`
C `readonly`
D `sealed`

---

4. Która z poniższych konstrukcji nie pasuje do pozostałych, aby było można zastosować przeciążenie metody?

A `public string Opisz() { ... }`

B `public string Opisz(string nazwa) { ... }`

C `public string Opisz(int liczba) { ... }`

D `public void Opisz() { ... }`

---

5. W klasie `Samochod` jest zadeklarowane pole `marka` typu `string`. Do klasy podpięto interfejs `IComparable<Samochod>`. W jaki sposób powinno się zaimplementować metodę `CompareTo` aby, sortowanie odbywało się odwrotnie alfabetycznie (Z-A)?

A

```
public int CompareTo(Samochod inny)
{
    return this.marka.CompareTo(inny.marka);
}
```

B

```
public int CompareTo(object obj)
{
    Samochod inny = obj as Samochod;
    return this.marka.CompareTo(inny.marka);
}
```

C

```
public int CompareTo(object obj)
{
    Samochod inny = obj as Samochod;
    return inny.marka.CompareTo(this.marka);
}
```

D

```
public int CompareTo(Samochod inny)
{
    return inny.marka.CompareTo(this.marka);
}
```

6. Która z poniższych kolekcji nie jest generyczna?

A `ArrayList`
B `List`
C `Queue`
A `Stack`

---

7. W programie zdefiniowany następujący typ wyliczeniowy:

```
enum DniTygodnia { Poniedziałek=2, Wtorek, Środa,
           Czwartek, Piątek, Sobota=3, Niedziela };
```

Jaką wartość ma `Niedziela` po rzutowaniu na typ `int`?

A 4
B 7
C 8
D 1

---

8. Która z poniższych metod pozwala usunąć i zwrócić ostatni element ze stosu?

A `Peek`
B `Enqueue`
C `Dequeue`
D `Pop`

---

9. Jakie słowo kluczowe należy użyć do nadpisania (ukrycia) metody?

A `new`
B `override`
C `virtual`
D `void`

---

10. Klasa `Osoba` jest abstrakcyjna. Która z poniższych konstrukcji nie jest poprawna?

A `Osoba student = new Student();`
B `Osoba student = new Osoba();`
C `List<Osoba> osoby = new List<Osoba>();`
D `Osoba belfer = new Nauczyciel();`

---

11. Którą z właściwości klasy `System.Exception` należy użyć do otrzymania komunikatu błędu?

A `Data`
B `HelpLink`
C `Message`
D `Source`

---

12. Którą z poniższych instrukcji można użyć do otrzymania właściwości tylko do zapisu?

A `public int Wiek { get; set; }`
B `public int Wiek { set; }`
C

```
private int wiek;
public int Wiek
{
    set { wiek = value; }
}
```

D

```
private int wiek;
public int Wiek
{
    get { return wiek; }
}
```

---

13. Która z instrukcji może posłużyć do deklaracji metody wirtualnej?

A `public abstract void Metoda() { ... }`
B `public virtual void Metoda() { ... }`
C `public virtual void Metoda();`
D `public abstract void Metoda();`

---

14. Która z instrukcji może posłużyć do deklaracji metody abstrakcyjnej?

A `public abstract void Metoda() { ... }`
B `public virtual void Metoda() { ... }`
C `public virtual void Metoda();`
D `public abstract void Metoda();`

Przeanalizuj kod:

```
public class A
{
    public void Metoda1() { Console.WriteLine("A
       Metoda1()"); }
    public virtual void Metoda2() {
       Console.WriteLine("A Metoda2()"); Metoda3(); }
    public virtual void Metoda3() {
       Console.WriteLine("A Metoda3()"); }
}

public class B : A
{
    public void Metoda1() { Console.WriteLine("B
       Metoda1()"); }
    public virtual void Metoda2() {
       Console.WriteLine("B Metoda2()"); Metoda1(); }
    public override void Metoda3() {
       Console.WriteLine("B Metoda3()"); }
}

public class C : B
{
    public override void Metoda2() {
       Console.WriteLine("C Metoda2()"); }
}

public class D : C
{
    public void Metoda1() { Console.WriteLine("D
       Metoda1()"); }
    public override void Metoda2() {
       Console.WriteLine("D Metoda2()"); Metoda2(); }
    public override void Metoda3() {
       Console.WriteLine("D Metoda3()"); }
}
```

15. Co będzie wywołane na konsoli po wykonaniu poniższej instrukcji?

```
A x1 = new B(); x1.Metoda2();
```

A

```
A Metoda2()
B Metoda3()
```

B `B Metoda2()`
C `B Metoda3()`
D `A Metoda2()`

---

16. Co będzie wywołane na konsoli po wykonaniu poniższej instrukcji?

```
B x2 = new D(); x2.Metoda3();
```

A `D Metoda3()`
B `B Metoda3()`
C będzie wyrzucony wyjątek
D `C Metoda3()`

---

17. Co będzie wywołane na konsoli po wykonaniu poniższej instrukcji?

```
C x3 = new C(); x3.Metoda1();
```

A `C Metoda2()`
B `C Metoda1()`
C będzie wyrzucony wyjątek
D `B Metoda1()`

---

18. Co będzie wywołane na konsoli po wykonaniu poniższej instrukcji?

```
A x4 = new D(); x4.Metoda1();
```

A `D Metoda1()`
B `B Metoda1()`
C będzie wyrzucony wyjątek
D `A Metoda1()`