Zad.28. Utwórz klasę, która implementuje oba interfejsy:

```
public interface A {
  public void pisz();
}
```

```
public interface B {
  public void pisz();
}
```

Zad.29. Utwórz klasę, która implementuje oba interfejsy:

```
public interface A {
  public void pisz(int liczba);
}
```

```
public interface B {
  public void pisz(float liczba);
}
```

Zad.30. Czy można utworzyć klasę, która implementuje oba interfejsy?

```
public interface A {
  public void pisz();
}
```

```
public interface B {
  public int pisz();
}
```

Zad.31. Utwórz klasę, która implementuje interfejs C:

```
public interface A {
  public void f();
}
```

```
public interface B {
  public void g();
}
```

```
public interface C extends A, B {
  public void h();
}
```

Zad.32. Oceń poprawność dziedziczenia interfejsów

```
public interface A {
  public int f();
}
```

```
public interface B {
  public float f();
}
```

```
public interface C extends A, B {
  public void h();
}
```

Wyjątki

Otwórz stronę `https://docs.oracle.com/javase/8/docs/api/index.html`.
Znajdź pakiet java.lang. i zapoznaj się z różnymi wyjątkami w Exception Summary.

Zad.33. Popraw poniższy kod, przechwytując pojawiający się tam wyjątek i wypisując odpowiedni komunikat.

```
public class Tablica{
  public static void main(String args[]) {
    int tab[] = new int[10];
    tab[10] = 5;
    System.out.println("Dziesiaty element tablicy ma wartosc: " + tab[10]);
}}
```

```
public class Tablica2{
  private int[] tablica = new int[10];
  public int pobierzElement(int indeks) {
    return tablica[indeks];
  }
  public void ustawElement(int indeks, int wartosc) {
    tablica[indeks] = wartosc;
}}
```

```
public class Main{
  public static void main(String args[]) {
    Tablica2 tablica = new Tablica2();
    tablica.ustawElement(5,10);
    int liczba = tablica.pobierzElement(10);
    System.out.println(liczba);
}}
```

Zad.34. Spróbuj skompilować poniższy program. Zaobserwuj komunikat kompilatora i popraw kod.

```
public class Hierarchia{
  public static void main(String args[]) {
    try
    {
      int liczba = 10/0;
    }
    catch(RuntimeException blad)
    {
       System.out.println("Wystapil blad: " + blad);
    }
    catch(ArithmeticException blad)
    {
       System.out.println("Wystapil blad: " + blad);
    }
  }
}
```

Zad.35. Skompiluj poniższy program. Zmodyfikuj kod w taki sposób, aby zgłoszone zostały oba typy błędów: ArithmeticException oraz NullPointerException.

```
public class Punkt{
  int x, y;
}
```

```
public class DwaWyjatki {
  public static void main(String args[]) {
    Punkt punkt = null;
    int liczba = 0;
    try{
      liczba = 10/0;
      punkt.x = liczba;
    }
    catch(ArithmeticException blad) {
      System.out.println("Nieprawidlowa operacja arytmetyczna");
      System.out.println(blad);
    }
    catch(RuntimeException blad) {
      System.out.println("Blad ogolny");
      System.out.println(blad);
    }
  }
}
```

Zad.36. Utwórz klasę z metodą main, która zgłasza obiekt klasy `Exception` wewnątrz bloku `try`. Przekaż parametr tekstowy konstruktorowi `Exception`. Przechwyć wyjątek wewnątrz sekcji `catch` i za pomocą metody `getMessage()` wypisz przekazany tekst.

Zad.37. Utwórz własną klasę wyjątków. Napisz dla tej klasy konstruktor przyjmujący parametr typu String i zapamiętujący ten parametr wewnątrz obiektu. Napisz metodę wyświetlającą ten łańcuch. Utwórz blok `try-catch` i wypróbuj ten wyjątek.

Zad.38. Napisz klasę z metodą, która zgłasza wyjątek stworzony w poprzednim zadaniu. Spróbuj ją skompilować bez specyfikacji wyjątku. Zaobserwuj komunikaty kompilatora. Dodaj odpowiednią specyfikację wyjątku. Wypróbuj swoją klasę i jej wyjątki wewnątrz bloku `try-catch`.

Zad.39. Napisz klasę wyjątku o nazwie Ujemna oraz klasę Pierwiastek, która będzie z niego korzystać. W klasie Piewiastek napisz metodę `oblicz()`, która generuje losowo dwie liczby i oblicza pierwiastek z ich różnicy. W przypadku, gdyby wynik odejmowania był ujemny, powinien zostać zgłoszony wyjątek Ujemna. Przetestuj działanie metody w main.