1. Klasy abstrakcyjne i interfejsy.

Zwykle po to tworzymy klasy by tworzyć ich egzemplarze. Okazuje się jednak często, że definiujemy klasy, które z założenia nie będą nigdy miały swoich egzemplarzy. Takie klasy nazywamy *klasami abstrakcyjnymi*. Wbrew temu, co mogłoby się wydawać na pierwszy rzut oka, te klasy pełnią bardzo ważną rolę przy projektowaniu hierarchii klas. Pozwalają bowiem na wyabstrahowanie wspólnych cech wielu definiowanych klas i jawne wskazanie, że wszystkie dziedziczące klasy muszą te cechy posiadać.

Cechy klasy abstrakcyjnej:

- nie można utworzyć obiektu tej klasy
- może zawierać niekompletną implementację (sama deklaracja metod)
- nieabstrakcyjna klasa pochodna musi implementować metody abstrakcyjne (inaczej sama również będzie abstrakcyjna).

```
abstract class Samochod {
	public void hamuje() {
	}
	abstract public void trabi(); //deklaracja metody
}
class Limuzyna extends Samochod {
	public void trabi() { //implementacja metody
	}
}
```

Czasami w klasie abstrakcyjnej chcemy opisać tylko interfejs, bez żadnych danych ani implementacji metod. Ponieważ takie wyabstrahowanie samego interfejsu jest bardzo ważne w Javie nadano mu specjalną postać składniową i nazwano ją *interfejsem*.

Cechy interfejsu:

- elementy publiczne
- pola jako stałe (static final) – konieczna inicjalizacja
- deklaracje metod

```
public interface Interfejs {
	public static final int INTER = 1;
	int FEJS = 2; // automatycznie public static final

	// deklaracja metody
	public void metoda();

}
```

Można implementować wiele interfejsów

```
interface A {
}

interface B {
}

interface C extends A {
} // rozszezanie interfejsu

class X implements B, C {
} // implementacja
```

Rozwiązanie problem braku wielodziedziczenia.

2. Klasa wewnętrzna i anonimowa.

- klasa wewnętrzna może być private oraz protected (zewnętrzna może być tylko public lub dostęp pakietowy)
- ma dostęp do wszystkich elementów klasy zewnętrznej (nawet tych prywatnych)
- rozwiązuje problem braku wielodziedziczenia

```
class A {
}

abstract class Abs {
}

class Y extends A {
      class I extends Abs {
      }
}
```

- tworzenie obiektu klasy wewnętrznej

```
Y y = new Y();
Y.I i = y.new I();
```

Wewnętrzna klasa statyczna:

- nie potrzebuje obiektu zewnętrznego
- brak dostępu do niestatycznych elementów klasy zewnętrznej

Klasa anonimowa:

- implementacja interfejsów i klas abstrakcyjnych

```
class Z extends D {
      E make() {
            return new E() { };
      }
}
```

Zadanie:

Zaimplementować aplikację za pomocą interfejsów i klas abstrakcyjnych. Klasy wewnętrznej lub anonimowej.