1. Polimorfizm. Rzutowanie.

Oprócz abstrakcji danych i dziedziczenia **polimorfizm** jest kolejnym podstawowym składnikiem języka programowania zorientowanego obiektowo. Dzięki **polimorfizmowi** uzyskujemy kolejną metodę separacji interfejsu od implementacji, pozwalającą na polepszenie organizacji i czytelności kodu oraz pozwalającej na tworzenie *rozszerzalnych* programów. Tworzone programów z wykorzystaniem polimorfizmu pozwala na ich rozwijanie nie tylko podczas powstawania ale również na późniejszym etapie kiedy okaże się że potrzebne jest rozszerzenie go o nowe możliwości.

Hermetyzcja umożliwia tworzenie nowych typów danych dzięki połączeniu charakterystyki z określonym zestawem zachowań. Pozwala ona na ukrywanie implementacji, oddzielając ją od interfejsu poprzez uczynienie szczegółów prywatnymi, polimorfizm zajmuje się natomiast oddzieleniem w kategoriach typów.

Traktowanie referencji do obiektu jak referencji do typu bazowego nazywane jest *rzutowaniem w górę* (upcasting) z powodu sposobu, w jaki rysuje się diagramy przedstawiające drzewa dziedziczenia – z klasą bazową u góry.

Poniżej przedstawiony został przykład rzutowania w górę. W poniższym przykładzie obiekty klasy Wind reprezentują pewien typ instrumentów , a zatem klasa Wind jest klasą pochodną Instrument:

```
//Typ wyliczeniowy Note
public enum Note {
	MIDDLE_C, C_SHARB, B_FLAT; //itd.
}


//klasa bazowa Instrument
public class Instrument {

	public void play(Note n)
	{
		System.out.println("Instrument.play()");
	}
}
```

```
//Klasa potomna Wind
public class Wind extends Instrument{
	public void play(Note n)
	{
		System.out.println("Wind.play() " + n );
	}
}


//Wywołanie metod z utworzonych klas
public class Music {
	public static void tune(Instrument i)
	{
		i.play(Note.MIDDLE_C);
	}

	public static void main(String[] args)
	{
		Wind flute = new Wind();
		tune(flute); //Rzutowanie w górę
	}
}
```

Można zaobserwować tutaj metodę tune(), do której przekazywana jest referencja typu Wind, przy czym rzutowanie nie jest koniecznym, ale jest dopuszczalne, ponieważ klasa Wind musi zawierać interfejs Instrument z którego dziedziczy. Upcasting może powodować „zawężenie” interfejsu klasy, nie może jednak uczynić go mniejszych od pełnego zakresu klasy bazowej.

Poniżej podany został przykład Polimorfizmu na podstawie klasy Figura i dziedziczącej z nich.

```
package ddorota;

public class Figura {

	void rysuj()
	{
		System.out.println("Figura.rysuj()");
	}

	void wymarz()
	{
		System.out.println("Figura.rysuj()");
	}
}
```

```
package ddorota;

public class Kolo extends Figura
{
	void rysuj()
	{
		System.out.println("Koło.rysuj()");
	}

	void wymarz()
	{
		System.out.println("Kolo.wymarz()");
	}
}
```

```
package ddorota;

public class Kwadrat extends Figura
{
	void rysuj()
	{
		System.out.println("Kwadrat.rysuj()");
	}

	void wymarz()
	{
		System.out.println("Kwadrat.rysuj()");
	}
}
```

```
package ddorota;

public class Trojkat extends  Figura
{
	void rysuj()
	{
		System.out.println("Trojkat.rysuj()");
	}

	void wymarz()
	{
		System.out.println("Trojkat.rysuj()");
	}
}
```

```
package ddorota;

public class CadApplication {

	public static void main(String[] args)
	{
		Figura f = new Kolo();
		f.rysuj();
		f.wymarz();
	}
}
```

Zadanie:

Zaimplementować aplikację wykorzystującą polimorfizm oraz mechanizm upcastingu.