Lista obejmuje następujące zagadnienia:

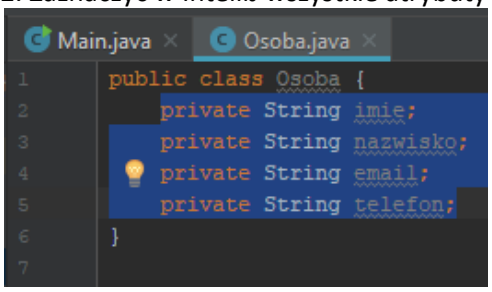
- dziedziczenie
- automatyczne tworzenie metod „wyświetl i ustaw” (ang. getters and setters)
- konstruktory, automatyczne generowanie konstruktrów, przeciążanie
- kolekcje elementów ArrayList
- operatory lambda
- interfejsy
- wzorzec projektowy Kompozyt

Kod można wpisywać w dowolnym *Java IDE*, natomiast zawarte tu wskazówki, jak ten proces efektywnie zautomatyzować, dotyczą *IDE IntelliJ Community Edition*

1. Utworzyć klasę Osoba:

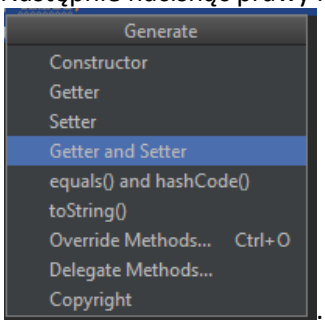
```
public class Osoba {
   private String imie;
   private String nazwisko;
   private String email;
   private String telefon;
}
```

2. Zaznaczyć w IntelliJ wszystkie atrybuty powyższej klasy

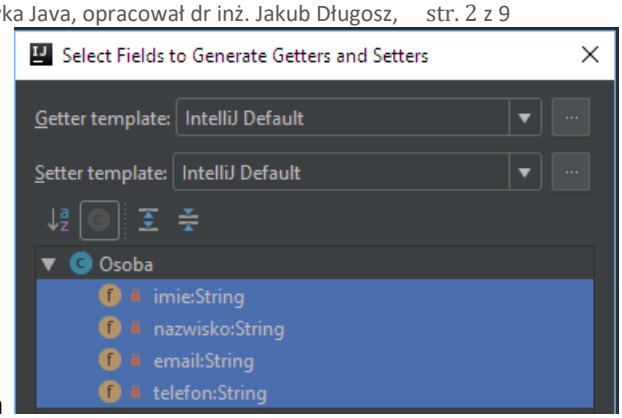


Następnie nacisnąć prawy klawisz myszy i wybrać 

, a później



.

Zaznaczyć wszystkie cztery pola 

i kliknąć OK.

Kod klasy Osoba powinien teraz wyglądać następująco:

```
public class Osoba {
    private String imie;
    private String nazwisko;

    public String getImie() {
        return imie;
    }

    public void setImie(String imie) {
        this.imie = imie;
    }

    public String getNazwisko() {
        return nazwisko;
    }

    public void setNazwisko(String nazwisko) {
        this.nazwisko = nazwisko;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getTelefon() {
        return telefon;
    }

    public void setTelefon(String telefon) {
        this.telefon = telefon;
    }

    private String email;
    private String telefon;
}
```

3. Przenieść dwie linie kodu:

```
private String email;
private String telefon;
```

pod linię:

```
private String nazwisko;
```

4. Umieścić kursor wewnątrz klasy *Osoba*, przed pierwszą metodą *getImie()*. Wcisnąć prawy klawisz myszy i wybrać *„Generate …”*, następnie *„Constructor”*, następnie zaznaczyć dwa pola *„imie:String”*, *„nazwisko:String”* i kliknąć OK.

Powinien zostać dodany następujący fragment kodu:

```
public Osoba(String imie, String nazwisko) {
    this.imie = imie;
    this.nazwisko = nazwisko;
}
```

5. Wykonać polecenia jak w kroku 4., tylko tym razem zaznaczyć trzy pola *„imie:String”*, *„nazwisko:String”*, *„email:String”* i kliknąć OK.

Powinien zostać dodany następujący fragment kodu:

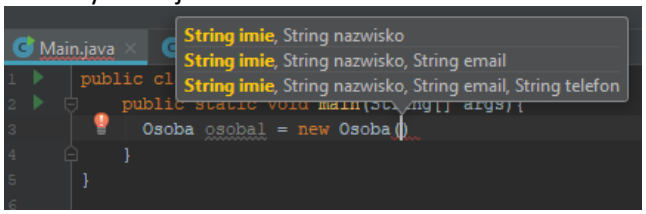
```
public Osoba(String imie, String nazwisko, String email) {
    this.imie = imie;
    this.nazwisko = nazwisko;
    this.email = email;
}
```

6. Wykonać polecenia jak w kroku 4., tylko tym razem zaznaczyć trzy pola *„imie:String”*, *„nazwisko:String”*, *„email:String”*, *„telefon:String”* i kliknąć OK.

Powinien zostać dodany następujący fragment kodu:

```
public Osoba(String imie, String nazwisko, String email, String telefon) {
    this.imie = imie;
    this.nazwisko = nazwisko;
    this.email = email;
    this.telefon = telefon;
}
```

7. Wewnątrz klasy Main rozpocząć wpisywanie kodu: `Osoba osoba1 = new Osoba() .` Edytor podpowie nam jak możemy zainicjować obiekt:



.

Utworzyć różne instancje obiektów klasy Osoba oraz posłużyć się dostępnymi dla nich metodami wyświetlając ich wynik w konsoli, np. jak w poniższym kodzie:

```
public class Main {
    public static void main(String[] args){
```

```
    Osoba osoba1 = new Osoba("Anna","Kowalska");
    Osoba osoba2 = new Osoba ("Tadeusz","Malinowski", "t.malinowski@blabla.pl");
    Osoba osoba3 = new Osoba ("Jan","Twardowski","j.twardowski@portal.pl","600 100 300");
    System.out.println("Nazwisko osoby pierwszej to: " + osoba1.getNazwisko());
    System.out.println("Mail osoby drugiej to: " + osoba2.getEmail());
    System.out.println("Telefon osoby trzeciej to: " + osoba3.getTelefon());
  }
}
```

Uruchomić program i zobaczyć na komunikaty w konsoli:

```
"F:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
Nazwisko osoby pierwszej to: Kowalska
Mail osoby drugiej to: t.malinowski@blabla.pl
Telefon osoby trzeciej to: 600 100 300

Process finished with exit code 0
```

8. Umieścić cały kod, który jest wewnątrz metody *main* klasy *Main* (opisany w punkcie 7.), w komentarzu. W tym celu zaznaczyć go, następnie wybrać z menu „Code" | „Comment with Block Comment (Ctrl+Shift+/)".

Zaimportować do pliku *Main.java* następujące biblioteki:

```
import java.util.ArrayList;
import java.util.List;
```

W tym celu wystarczy umieścić kursor wewnątrz słowa *List* oraz wewnątrz słowa *ArrayList* i nacisnąć *alt+Enter*.

Dodać do metody *main* klasy *Main* następujący kod:

```
    List<Osoba> listaOsob = new ArrayList<>();
    listaOsob.add(new Osoba("Zygmunt","Malanowski","tajemniczy92@portal.pl"));
    listaOsob.add(new Osoba("Krzysztof","Dobrzynski","dobrzynski@dobrzynski.pl","792 792 790"));
    listaOsob.add(new Osoba("Andrzej","Mazur"));

    for(Osoba osoba : listaOsob){
      System.out.println("Imię: "+osoba.getImie()+" , nazwisko: "+osoba.getNazwisko());
    }
```

Uruchomić program. W konsoli powinno pojawić się:

```
"F:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
Imię: Zygmunt , nazwisko: Malanowski
Imię: Krzysztof , nazwisko: Dobrzynski
Imię: Andrzej , nazwisko: Mazur

Process finished with exit code 0
```

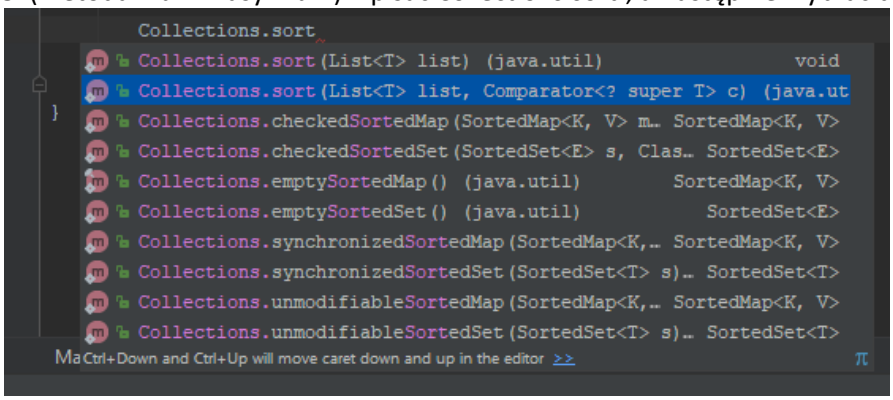
9. Zamienić pętlę for z punktu 8. na następujące wyrażenie, wykorzystujące **wyrażenia lambda**:

```
    listaOsob.forEach(
        osoba -> System.out.println("Imię: "+osoba.getImie()+" , nazwisko: "+osoba.getNazwisko())
    );
```

Uwaga: operator **->** oznacza wyrażenie lambda

Zwrócić uwagę, że w wierszu edytora, w którym znalazło się wyrażenie lambda, pojawiła się ikona  .

10. Posortować listę osób utworzoną w punkcie 8. rosnąco względem nazwiska. W tym celu na końcu kodu z punktu 8. (metoda *main* Klasy *Main*) wpisać *Collections.sort* , a następnie wybrać drugi element z rozwiniętej listy :



Wpisać kod:

```
Collections.sort(listaOsob, Comparator.comparing(Osoba::getNazwisko));
```

Ponownie wyświetlić listę osób, powinna teraz być posortowana:

```
Imię: Krzysztof , nazwisko: Dobrzynski
Imię: Zygmunt , nazwisko: Malanowski
Imię: Andrzej , nazwisko: Mazur
```

11. Zamienić kod z punktu 10. :

```
Collections.sort(listaOsob, Comparator.comparing(Osoba::getNazwisko));
```

na **wyrażenie lambda**:

```
Collections.sort(listaOsob, Comparator.comparing(osoba -> osoba.getNazwisko()));
```

12. Utworzyć klasę *Pracownik* rozszerzającą (dziedziczącą) z klasy *Osoba*:

```
public class Pracownik extends Osoba {
 private String stanowisko;
}
```

Dla zmiennej *stanowisko* utworzyć metody ”wyświetl i ustaw” (getter and setter) powtarzając krok z punktu 2.

13. Czy można w metodzie *main* klasy *Main* użyć poniższej inicjalizacji?

```
Pracownik prac1 = new Pracownik("Anna","Kowalska");
```

A czy można użyć poniższej inicjalizacji?

```
Pracownik prac1 = new Osoba("Anna","Kowalska");
```

14. W klasie Pracownik utworzyć następujące konstruktory:

```
public Pracownik(String imie, String nazwisko) {
   super(imie, nazwisko);
}
```

```
  public Pracownik(String imie, String nazwisko, String stanowisko) {
    super(imie, nazwisko);
    this.stanowisko = stanowisko;
  }

  public Pracownik(String imie, String nazwisko, String email, String stanowisko) {
    super(imie, nazwisko, email);
    this.stanowisko = stanowisko;
  }

  public Pracownik(String imie, String nazwisko, String email, String telefon, String stanowisko) {
    super(imie, nazwisko, email, telefon);
    this.stanowisko = stanowisko;
  }
```

Uwaga: słowo kluczowe ***super*** oznacza odwołanie do klasy bazowej

15. W metodzie *main* klasy *Main* zainicjować kilka obiektów klasy *Pracownik* i wyświetlić ich atrybuty w konsoli.

16. W pliku Main.java dodać następujący interfejs ocenPracownika:

```
interface ocenPracownika {
  boolean wpiszPochwale(String tekst);
  boolean wpiszNagane(String tekst);
}
```

Uwaga: Zwrócić uwagę na brak nawiasów klamrowych {} w sygnaturze metod *wpiszPochwale(String tekst)* i *wpiszNagane(String tekst)*. Co by się stało, gdyby je umieścić?

17. Co wymusza na klasie *Pracownik* dodanie do jej definicji wyrażenia *implements ocenPracownika* ?

```
public class Pracownik extends Osoba implements ocenPracownika {
 ...
}
```

18. Zapoznać się ze **wzorcem projektowym Kompozyt** (ang. Composite)
https://pl.wikipedia.org/wiki/Kompozyt_(wzorzec_projektowy)

19. Zaimplementować kod ze strony http://czub.info/2015/wzorzec-composite-kompozyt/:

kod:

```
import java.util.ArrayList;

//
```

```
// Definicja interfejsu komponentu
//
interface ComponentInterface{
   void process();
}

//
// Klasa liscia - definiuje objekty które nie mają dzieci
//
class Leaf implements ComponentInterface{

   @Override
   public void process(){
      System.out.println("Process: " + this);
   }
}//koniec class Leaf


// klasa Main
class Main{
   public static void main(String[] args){
      DesignPatternsComposite dpCom = new DesignPatternsComposite();
      dpCom.test();
   }
}

//
// Klasa kompozytu
//
class Composite implements ComponentInterface{

   private ArrayList< ComponentInterface > children = new ArrayList< ComponentInterface >();

   public void addComponent(ComponentInterface child){
      System.out.println("Add children: " + child);
      children.add(child);
   }

   @Override
   public void process(){
      if (!children.isEmpty()){
         for (ComponentInterface child : children){
            if (child instanceof ComponentInterface){
               child.process();
            }
         }

      }
   }//koniec public void process(){
}//koniec class Composite
```

```
//
// Klasa klienta:
// tworzymy podstawowe obiekty, jeden kompozyt będzie zawierał dwa
// takie same obiekty liści
// testowo wykonujemy takie same operacje na stworzonych obiektach,
// widać ze niezależnie od punktu wejścia (liść, kompozyt czy cała struktura),
// wszystkie opracje wykonują się bez problemu
//
public class DesignPatternsComposite{

   public static void main(String[] args){
      DesignPatternsComposite dpCom = new DesignPatternsComposite();
      dpCom.test();
   }

   public void test(){

      System.out.println("Add items ...");

      Leaf leaf = new Leaf();
      Leaf leaf2 = new Leaf();
      Leaf leaf3 = new Leaf();

      Composite childComposite = new Composite();
      Composite childComposite2 = new Composite();

      childComposite.addComponent(leaf);

      childComposite2.addComponent(leaf);
      childComposite2.addComponent(leaf2);
      childComposite2.addComponent(leaf3);

      Composite composite = new Composite();
      composite.addComponent(childComposite);
      composite.addComponent(childComposite2);

      System.out.println("Execute process on Leaf ...");

      leaf.process();
      leaf2.process();

      System.out.println("Execute process on Children ...");

      childComposite.process();
      childComposite2.process();

      System.out.println("Execute process on Composite ...");

      composite.process();
   }//koniec public void test()
} // koniec klasy DesignPatternsComposite
```

Wynik działania kodu powinien być następujący:

```
"F:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
Add items ...
Add children: Leaf@154617c
Add children: Leaf@154617c
Add children: Leaf@a14482
Add children: Leaf@140e19d
Add children: Composite@17327b6
Add children: Composite@14ae5a5
Execute process on Leaf ...
Process: Leaf@154617c
Process: Leaf@a14482
Execute process on Children ...
Process: Leaf@154617c
Process: Leaf@154617c
Process: Leaf@a14482
Process: Leaf@140e19d
```

```
Execute process on Composite ...
Process: Leaf@154617c
Process: Leaf@154617c
Process: Leaf@a14482
Process: Leaf@140e19d

Process finished with exit code 0
```