1. Dziedziczenie metod bez przedefiniowania i z przedefiniowaniem: rola słowa kluczowego super; przeciążanie metod.

```
class Osoba1 {
  int wiek;
  String nazwisko;

  public Osoba1(int wiek_, String nazwisko_) {
    wiek = wiek_;
    nazwisko = new String(nazwisko_); }

  @Override
  public String toString() {
    return "wiek= " + wiek + ", nazwisko= " + nazwisko; }

  public int getPobory() {      return -1;   }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)    {          return true;      }
    if (obj == null)    {          return false;     }
    Osoba1 other = (Osoba1) obj;
    if (this.wiek != other.wiek) { return false; }
    if (!Objects.equals(this.nazwisko, other.nazwisko))
                        {          return false;  }
    if (getPobory() != other.getPobory()) {   return false; }
    return true;
  }
  public String pokaz() {  return toString();   }
}
class Osoba2 extends Osoba1
 { int pobory;
  public Osoba2(int wiek_, String nazwisko_, int pobory_)  {/*   */}
  @Override
  public String toString()                {/*   */}
  @Override
   public int getPobory()                 {/*   */ }
   public String pokaz(String napis)      {/*   */}
 }
```

/*Zdefiniuj konstruktor, który dziedziczy po Osoba1 i przypisuje pobory_ do składowej pobory oraz metody: ***toString,*** która tworzy i zwraca łańcuch zawierający: wiek, i nazwisko za pomocą dziedziczonej metody toString() oraz pobory; ***pokaz*** wraz przekazanym przez nagłówek łańcuchem zawierającym menu – w definicji metody należy wykorzystać metodę toString; **getPobory**, która zwraca wartość atrybutu pobory. **Należy wyjaśnić, dlaczego wystarczy metoda equals w klasie Osoba1, aby poprawnie porównać dowolną parę obiektów typu Osoba1 i Osoba2.***/

```
public class GUI_1 {
  public static void main(String[] args) {
    Osoba2 p2 = new Osoba2(20, "Kowalski", 5);
    Osoba1 p1 = new Osoba1(15, "Nowak");
    System.out.println(p2.toString());
        //wiek= 20, nazwisko= Kowalski, pobory= 5
   System.out.println(p2.pokaz("Dane osoby: "));
       //Dane osoby: wiek= 20, nazwisko= Kowalski, pobory= 5
    System.out.println(p1.toString()); //wiek= 15, nazwisko= Nowak
    System.out.println(p1.pokaz());    //wiek= 15, nazwisko= Nowak
    System.out.println(p1.equals(p2)); //false
    System.out.println(p2.equals(p1)); //false
    System.out.println(p2.equals(p2)); //true
    System.out.println(p1.equals(p1)); //true
  } }
```

---

2. Polimorfizm czyli korzystanie z metod przedefiniowanych

```
interface Dane
 {  String produkt =" Typ osoby: ";
public  int podaj_typ(); }

class Osoba1 implements Dane, Comparable{
  int wiek;
  String nazwisko;
  public Osoba1(int wiek_, String nazwisko_) {
    wiek = wiek_;
    nazwisko = new String(nazwisko_); }
  @Override
  public String toString() {
    return "wiek= " + wiek + ", nazwisko= " + nazwisko; }
  public int getPobory() {      return -1;   }
  @Override
  public boolean equals(Object obj) {
    Osoba1 other = (Osoba1) obj;
    if (this.wiek != other.wiek) {          return false;      }
    if (!Objects.equals(this.nazwisko, other.nazwisko)) {
      return false;      }
    if (getPobory() != other.getPobory()) { return false;   }
    return true;   }
 public int podaj_typ()   {   return 1;  }
 @Override
  public int compareTo(Object o) {
    Osoba1 o1=(Osoba1)o;
    return nazwisko.compareTo(o1.nazwisko);   }
 }
class Osoba2 extends Osoba1 {
 int pobory;
 public Osoba2(int wiek_, String nazwisko_, int pobory_) { /*   */ }
@Override
  public String toString() { /*   */ }
@Override
  public int getPobory()  {      /*   */  }
@Override
  public int podaj_typ()   {     /*   */  }
}
```

/*Należy zdefiniować konstruktor inicjujący atrybuty obiektu, (zastosuj słowo super) i przypisuje pobory_ do składowej pobory oraz 3 metody: toString(), podaj_typ (), getPobory(). **Czy można zrezygnować z definicji konstruktora?**

Metoda toString() zwraca wartości atrybutów: wiek, nazwisko, za pomocą dziedziczonej metody toString() oraz pobory - do definicji należy wykorzystać słowo super; druga podaje typ osoby (implementacja metody podaj_typ(); trzecia zwraca wartość arybutu pobory. **Wyjaśnij, dlaczego, ninie ma potrzeby przedefiniowania metody equals w klasie Osoba2.** */

```
public class GUI_2 {
  static TreeSet<Osoba1> dane=new TreeSet();
  static ArrayList<String> model()
   {  ArrayList<String> model = new ArrayList();
     /* tutaj należy wstawić  kod tworzący kolekcję model*/
      return model; }
  static ArrayList<String> wyprowadz(Osoba1 p)
  {  dane.add(p);
     return /*…*/; }
  public static void main(String[] args) {
    Osoba2 p2 = new Osoba2(20, "Kowalski", 5);
    Osoba1 p1 = new Osoba1(15, "Nowak");
    System.out.println(p1.toString());   //wiek= 15, nazwisko= Nowak
    System.out.println(wyprowadz(p1));
                              //[wiek= 15, nazwisko= Nowak Typ osoby: 1]
     System.out.println(p2.toString());
                               // wiek= 20, nazwisko= Kowalski, pobory= 5
     System.out.println(wyprowadz(p2));
                  //  [wiek= 20, nazwisko= Kowalski, pobory= 5 Typ osoby: 2,
                   //   wiek= 15, nazwisko= Nowak Typ osoby: 1]
  }
}
```

3. Napisz szkielet programu (utworzenie okna i wykonanie obsługi zdarzeń typu **ActionListener** oraz **ItemListener**), który:

1) tworzy okno pokazane z lewej strony

1) wstawia z pola typu **JTextField** („**Dodaj tytul**”) dane typu String do kolekcji dane typu **TreeSet** metodą **add** po naciśnięciu klawisza „**Dodaj tytul**” i automatycznie wyświetla aktualną zawartość kolekcji w komponencie „**Slownik tytulow**” typu JComboBox
2) po naciśnięciu na wybraną pozycję listy komponentu „**Slownik tytulow**” wyswietla jej zawartość w polu „**Wybrany tytul**”