♣

Java

Semester	zimowy 2022/23
# ects	5
forma zaliczenia	egzamin
books	
Honors	☐
Professor	
Property	

Pytania na egzamin ustny

Wykłady 1/2

- **słowo kluczowe static**
  oznaczenie metody lub pola jako static informuje nas, że wiążą się one z całą klasą, wieloma jej instancjami, metody static nie mogą wywoływać niestatycznych
  Atrybuty statyczne inicjalizowane są w momencie załadowania klasy, a nie stworzenia instancji Wartość zmiennej static można ustawić ręcznie przy deklaracji, w bloku static , ale również w konstruktorze. Wtedy przy każdorazowym tworzeniu instancji danej klasy, będzie
  ustawiana/inkrementowana wartość takiej zmiennej.
- **klasa abstrakcyjna vs interfejs**
  klasa abstrakcyjna posiada conajmniej jedną metodę abstrakcyjną, czyli tylko zadeklarowaną a bez definicji, nie można tworzyć ich instancji
  interfejs natomiast składa się jedynie z niezdefiniowanych metod (to taki outline klasy), mogą one być argumentami i mogą być zwracane
- **Javadoc (komenda, co to i jakie komentarze uznaje)**

  Dokumentacja tworzona jest przez komentarze dokumentacyjne (documentation comments), w ktorych używany jest kod HTML
  `javadoc nazwaPliku.java`
  Komentarze dokumentacyjne ograniczone są znakami: `/**` i `*/`
  javadoc jest narzedziem, ktore generuje dokumentacje przegladając komentarze umieszczone w plikach źródłowych Java (lecz nie tylko). Domyślnie generowane są opisy metod publicznych i chronionych.
  Komentarz sklada sie z komentarza głównego oraz wystepującej po nim sekcji tag-ów.
  Sekcja tagów rozpoczyna się pierwszym tagiem blokowym. Tag blokowy zaczyna sie znakiem @, który rozpoczyna linie (nie licząc poprzedzających go białych znaków i gwiazdek). Jeśli nie ma wiodącej gwiazdki (*) to białe znaki nie są usuwane (pozwala to utrzymywać formatowanie w sekcjach <pre>)
  Tag to specjalne słowo kluczowe, które javadoc potrafi zrozumieć i "wykonać". Tagi dzielą się na blokowe (@tag) oraz wplecione ({@tag}).
  Tag blokowy musi pojawiać sie na początku linii. Znak @ użyty w tekście nie jest interpretowany jako rozpoczęcie tagu. Z drugiej strony, jeśli chcemy wyswietlić "małpe" na początku linii to trzeba zostosować kod: @
  Z tagiem blokowym powiązany jest fragment tekstu - ciągnie się on od tagu po tag następny lub po koniec całego bloku dokumentacji.
  Tagi wplecione mogą pojawić się wewnętrz tekstu.
  Na początku bloku dokumentacji umieszczane jest krótkie podsumowanie. Kończy się ono na pierwszej kropce, po której jest biały znak albo tag.
  Jesli pola deklarowane są w jednej instrukcji to komentarze są powielane.
  javadoc ma mechanizmy kopiowania komentarzy w przypadkach dziedziczenia.
- **modyfikatory dostępu**, wszedł na refleksyjne i dynamiczne - czy można dostać wartość prywatnej zmiennej? (Tak, zmieniając modyfikator dostępu albo parsując do XML i sobie stamtąd odczytać

public: dostęp skąd tylko chcemy
protected: dostęp z danej klasy, jej podklas i klas pakietu
default: dostęp tylko z danej klasy i pakietu
private: dostęp tylko z danej klasy

- **czym sa interfejsy, wszystko co o nich wiesz i wymień tyle ile ich pamietasz**
  znane interfejsy: runnable, callable, clonable, comparable, list/set/queue, supplier, consumer, function, predicate, serializable,

Wykład 3

- **opisać działanie i różnice 'throw' i 'throws'**. Czy 'throw' będzie działać w metodzie bez 'throws' (nie). Dodatkowo opowiedzieć o **Runtime Exception** i czy możemy obsługiwać takie wyjątki jak nullPointerException. (tak)
  Exceptions dzielimy na:
  -Checked, czyli przewidywalne: `FileNotFound, IOException, ClassNotFound, Interrupted`
  -Unchecked, czyli `RuntimeExceptions: NullPointer, IndexOutOfBonds,ClassCast`

- czym się różni **ArrayList od Vectora**, gdzie przechowują elementy, co jest szybsze
  Szybsze jest ArrayList bo nie jest synchronized
  ArrayList – jest to podstawowa implementacja listy w Javie. Można też ją określić jako samo-rozszerzalną tablicę, ponieważ jej implementacja bazuje na tablicy, która jest powiększana wraz ze wzrostem rozmiaru listy. Dzięki temu, jest to **najwydajniejsza implementacja listy w Javie** (w bibliotece standardowej).
  Charakteryzuje się szybkim losowym dostępem do poszczególnych elementów poprzez metodę `get(int index).`
  Vector – jest implementacją, bardzo podobny do `ArrayList`
  jednak z tą zasadniczą różnicą, że wszystkie jego operacje są synchronizowane. Co sprawia, że jest on bardziej odpowiedni do środowiska wielowątkowego.

- **HashTable i HashMap** , jak działają, co implementują / rozszerzają, i czym się różnią (trzeba wspomnieć że jedno może trzymać nulle drugie nie)
  Klasa Java Hashtable implementuje interfejsy Map, Cloneable i Serializable. Rozszerza klasę Dictionary.
  Klasa Java HashMap implementuje interfejsy Map, Cloneable i Serializable. Rozszerza klasę AbstractMap.
  • Hashtable jest podobny do HashMap w Javie. Najbardziej znaczącą różnicą jest to, że Hashtable jest synchronizowany, a HashMap nie. Dlatego Hashtable jest wolniejszy niż HashMap właśnie z powodu synchronizacji.
  • Poza problemem synchronizacji, Hashtable nie pozwala na użycie null jako wartości lub klucza. HashMap zezwala na jeden klucz i wiele wartości null.
  • Hashtable dziedziczy klasę Dictionary, podczas gdy HashMap dziedziczy klasę AbstractMap.
  • HashMap może być przemierzana przez Iterator. Hashtable może być przemierzana nie tylko przez Iterator, ale także Enumerator.

- wszystko o **HashTable**, jakie ma metody, jak pozyskać element

  put(Object key, Object value) i **get(Object key)**,
  public Set<K> **keySet()**;
  public synchronized Enumeration<K> **keys()**;
  public Collection<V> **values()**;
  public synchronized Enumeration<V> **elements**();
  public Set<Map.Entry<K,V>> **entrySet**();
  public synchronized V **remove**(Object key);

- **Podobieństwa i różnice między Vector i Hashtable, jak przeiterować po Hashtable**

  Using Enumeration Interface
  Using keySet() method of Map and Enhance for loop
  Using keySet() method of Map and Iterator Interface
  Using entrySet() method of Map and enhanced for loop
  Using entrySet() method of Map and Iterator interface
  Using Iterable.forEach() method from version Java 8
  Oba są synchronized, Vectors are ideal for storing lists of items where you typically do not need to search through the list for a specific item. Hashtables are ideal for storing key-value pairs. Access to any key-value is very fast when searching by key unlike Vectors which require a search through the entire Vector

- Różnica arraylist i hashSet - kiedy której bym użył, jakie są różnice, krótki opis też (mówcie wszystko co wiecie mi dał dodatkowe pkt za złożoności operacji na nich)
  Implementation: ArrayList implements List interface while HashSet implements Set interface in Java.
  Internal implementation: ArrayList is backed by an Array while HashSet is backed by an HashMap.

Duplicates: ArrayList allows duplicate values while HashSet doesn't allow duplicates values.
Constructor : ArrayList have three constructor which are ArrayList(), ArrayList(int capacity) ArrayList(int Collection c) while HashSet have four constructor which are HashSet(), HashSet(int capacity), HashSet(Collection c) and HashSet(int capacity, float loadFactor)
Ordering : ArrayList maintains the order of the object in which they are inserted while HashSet is an unordered collection and doesn't maintain any order.
Indexing : ArrayList is index based we can retrieve object by calling get(index) method or remove objects by calling remove(index) method while HashSet is completely object based. HashSet also does not provide get() method.
Null Object: ArrayList not apply any restriction, we can add any number of null value while HashSet allow one null value.

- **Collection, przykłady interfejsów rozszerzających. Vector opis metod. Metody wspólne w collection.**

  boolean **add**(Object o)
  void **clear**()
  boolean **contains**(Object o)
  boolean isEmpty()
  Iterator iterator()
  boolean **remove(**Object o)
  int **size**()
  Object[] **toArray**()
  Przykłady interfejsów rozszerzających: list, set, queue

- **Kolekcje z naciskiem na te z wykładu, kazał mi opisać TreeMap, złożoności, co jest lepsze w porównaniu z ArrayList a co gorsze**

  ArrayList
  Allows duplicates (it is a List)
  Amortized O(1) to add to the end of the list
  O(n) to insert anywhere else in the list
  O(1) to access
  O(n) to remove

  TreeMap //nastawiona na sortowanie
  Does not allow duplicate keys (it is a Map)
  O(logn) to insert
  O(logn) to access
  O(logn) to remove

- **Vector vecraw=new Vector,Vector<Object> vecobj = new Vector, porównaj, opisz jak działają**
  Tablica wie jakiego jest typu i nie przyjmuje dowolnych referencji. Ale uwaga: zbyt ogolny typ tablicy powoduje, ze kompilator nie jest w stanie uchronic nas przed problemami
  ZALETY oznaczania typu:
  -kontrola typów na poziomie kompilacji, a nie dopiero w trakcie działania programu
  -brak potrzeby rzutowania,
  -konstruowanie ogólnych algorytmów.

- **wszystkie metody iteracji przez tablicę/listę rozpisać - czyli iteratory, enumeratory, lambda, różne pętlę for**

```
// Simple For loop
        System.out.println("==============> 1. Simple For loop Example.");
        for (int i = 0; i < crunchifyList.size(); i++) {
            System.out.println(crunchifyList.get(i));
        }
// New Enhanced For loop
        System.out.println("\n==============> 2. New Enhanced For loop Example..");
        for (String temp : crunchifyList) {
            System.out.println(temp);
        }
// Iterator - Returns an iterator over the elements in this list in proper sequence.
        System.out.println("\n==============> 3. Iterator Example...");
        Iterator<String> crunchifyIterator = crunchifyList.iterator();
        while (crunchifyIterator.hasNext()) {
            System.out.println(crunchifyIterator.next());
        }
        // ListIterator - traverse a list of elements in either forward or backward order
        // An iterator for lists that allows the programmer to traverse the list in either direction, modify the list during iteration
        // and obtain the iterator's current position in the list.
        System.out.println("\n==============> 4. ListIterator Example...");
        ListIterator<String> crunchifyListIterator = crunchifyList.listIterator();
        while (crunchifyListIterator.hasNext()) {
            System.out.println(crunchifyListIterator.next());
        }
```

```
// while loop
        System.out.println("\n==============> 5. While Loop Example....");
        int i = 0;
        while (i < crunchifyList.size()) {
            System.out.println(crunchifyList.get(i));
            i++;
        }
// Iterable.forEach() util: Returns a sequential Stream with this collection as its source
        System.out.println("\n==============> 6. Iterable.forEach() Example....");
        crunchifyList.forEach((temp) -> {System.out.println(temp);
});
// collection Stream.forEach() util: Returns a sequential Stream with this collection as its source
        System.out.println("\n==============> 7. Stream.forEach() Example....");
        crunchifyList.stream().forEach((crunchifyTemp) -> System.out.println(crunchifyTemp));
    }
```

- **properties**

Properties to rozszerzenie tablicy haszującej nastawione na przechowywanie par Stringów:
public class Properties extends Hashtable<Object,Object>;
public synchronized Object setProperty(String key, String value);
public String getProperty(String key);

Klasa Properties "współpracuje" z plikami tekstowymi zapisanymi w określonym formacie,
`public synchronized void load( InputStream inStream) throws IOException`
`public void store( OutputStream out, String comments)throws IOException`
`public synchronized void loadFromXML( InputStream in)`
`throws IOException, InvalidPropertiesFormatException`
`public synchronized void storeToXML( OutputStream os, String comment)`
`throws IOException`

- Wypisać Properties do pliku

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Enumeration;
import java.util.Properties;

public class Main {
    public static void main(String[] args) throws IOException {
        Properties properties = new Properties();
        properties.load(new FileInputStream("file.properties"));
        Enumeration keys = properties.propertyNames();
        while (keys.hasMoreElements()) {
            String key = (String) keys.nextElement();
            String value = properties.getProperty(key);
            System.out.println(key + ": " + value);
        }
    }
}
```

- **Interfejs Clonable**

  W Javie do klonowania został stworzony interfejs `Cloneable`, którym oznacza się klasę, dla której ma być dozwolone klonowanie. Jednak nie jest to idealne rozwiązanie. **Interfejs ten nie zawiera żadnych metod,** a metoda `clone` w klasie `Object` jest `protected`, więc nie możemy wywołać metody `clone`, tylko dlatego, że klasa implementuje `Cloneable`.

  Interfejs ten zmienia tylko zachowanie metody `clone` w `Object` - jeśli klasa implementuje `Cloneable` to zwracana jest kopia pole po polu obiektu, a w przeciwnym wypadku rzucany jest wyjątek `CloneNotSupportedException`. Jest to trochę nietypowe wykorzystanie interfejsu — zamiast mówić klientowi co klasa może zrobić, zmienia zachowanie metody nadklasy i do niczego więcej się nie przyda.

  Z interfejsu to nie wynika, ale klasa implementująca interfejs `Cloneable` powinna dostarczyć poprawnie funkcjonująca metodę `clone`.

  ```
  @Override
      public Person clone() {
          try {
              return (Person) super.clone();
          } catch (CloneNotSupportedException e) {
              throw new Error("This should not happen!");
          }
      }
  ```

- **Interfejs Comparable**
  Do porównywania obiektów w Javie służy metoda `compareTo`

W przeciwieństwie do innych metod w tym rozdziale serii nie jest zadeklarowana w `Object`
Jest to osobna metoda zdefiniowana w interfejsie `Comparable`
Poza sprawdzaniem równości obiektów, tak jak w metodzie `equals`, oferuje jeszcze określanie, czy jest mniejszy lub większy niż inny obiekt. Implementując interfejs `Comparable` **określamy domyślny porządek** (ang. natural ordering) klasy, według którego m.in będzie układana w kolekcjach i na którym polega wiele algorytmów w Javie.

- Posortować wektor po długości stringów, z uzyciem comparable

```
@Override
    public int compareTo(String s) {
        return this.length() - s.length();
    }

class Main{
    public static void main(String[] args){
        List<String> lista = new ArrayList<>(Arrays.asList("x","","bb","aaa"));
        Collections.sort(lista);
        System.out.println(lista);
    }
}
```

- Napisać jakąś kolekcje np. hashmape i posortować klucze alfabetycznie

```
public class Sortowanie implements Comparable<Sortowanie>{
    public String text;
    Sortowanie(String s){
        this.text=s;
    }
    public static void main(String[] args) {
        Sortowanie[] doSortowania = {new Sortowanie("b"),new Sortowanie("a"), new Sortowanie("c")};
        Arrays.sort(doSortowania);
        for(Sortowanie s: doSortowania){
            System.out.println(s.text);
        }
    }
    @Override
    public int compareTo(Sortowanie o) {
        return this.text.compareTo(o.text);
    }
}
```

Wykład 4

- **Metody pomostowe**
  to metody stworzone przez kompilator, które implementują polimorfizm w rozszerzonych typach generycznych np. Box<T> ma metodę set(), która wg bytecodu przyjmuje obiekt, a set() w klasie potomnej myBox extends Box<String> przyjmuje String więc kompilator dodaje do myBox metodę `set(Object data){set((String)data);}`
- **Wildcard**

  Consider the problem of writing a routine that prints out all the elements in a collection. Here's how you might write it in an older version of the language (that is, a pre-5.0 release):

  ```
  void printCollection(Collection c) {
      Iterator i = c.iterator();
      for (k = 0; k < c.size(); k++) {
          System.out.println(i.next());
      }
  }
  ```

  And here is a naive attempt at writing it using generics (and the new `for` loop syntax):

  ```
  void printCollection(Collection<Object> c) {
      for (Object e : c) {
          System.out.println(e);
      }
  }
  ```

The problem is that this new version is much less useful than the old one. Whereas the old code could be called with any kind of collection as a parameter, the new code only takes `Collection<Object>`, which, as we've just demonstrated, is **not** a supertype of all kinds of collections!

So what **is** the supertype of all kinds of collections? It's written `Collection<?>` (pronounced "collection of unknown"), that is, a collection whose element type matches anything. It's called a **wildcard type** for obvious reasons. We can write:

```
void printCollection(Collection<?> c) {
    for (Object e : c) {
        System.out.println(e);
    }
}
```

and now, we can call it with any type of collection. Notice that inside `printCollection()`, we can still read elements from `c` and give them type `Object`. This is always safe, since whatever the actual type of the collection, it does contain objects. It isn't safe to add arbitrary objects to it however:

```
Collection<?> c = new ArrayList<String>();
c.add(new Object()); // Compile time error
```

Since we don't know what the element type of `c` stands for, we cannot add objects to it. The `add()` method takes arguments of type `E`, the element type of the collection. When the actual type parameter is `?`, it stands for some unknown type. Any parameter we pass to `add` would have to be a subtype of this unknown type. Since we don't know what type that is, we cannot pass anything in. The sole exception is `null`, which is a member of every type. On the other hand, given a `List<?>`, we **can** call `get()` and make use of the result. The result type is an unknown type, but we always know that it is an object. It is therefore safe to assign the result of `get()` to a variable of type `Object` or pass it as a parameter where the type `Object` is expected.

- **Typy Generyczne**
  In a nutshell, generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

  Code that uses generics has many benefits over non-generic code:

  - Stronger type checks at compile time.A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
  - Elimination of casts.The following code snippet without generics requires casting:When re-written to use generics, the code does not require casting:

    ```
    List list = new ArrayList();
    list.add("hello");
    String s =  (String)  list.get(0);
    ```

    ```
    List<String> list = new ArrayList<String>();
    list.add("hello");
    String s = list.get(0);   // no cast
    ```
  - Enabling programmers to implement generic algorithms.By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

- **Typ surowy**

  A *raw type* is the name of a generic class or interface without any type arguments. For example, given the generic Box class:

  ```
  public class Box<T> {
     public void set(T t) { /* ... */ }
     // ...}
  ```

  To create a parameterized type of Box<T>, you supply an actual type argument for the formal type parameter T:

  ```
  Box<Integer> intBox = new Box<>();
  ```

  If the actual type argument is omitted, you create a raw type of Box<T>:

  ```
  Box rawBox = new Box();
  ```

  Therefore, Box is the raw type of the generic type Box<T>. However, a non-generic class or interface type is *not* a raw type.

- **typy generyczne vs wildcard**
- **ograniczenia w generics**

-nie można używać typów prymitywnych (Box<int>),
-nie można używać operatora new (E e = new E();),
-nie można deklarować typów statycznych (private static T v;)
-typów parametryzowanych (List<Integer>) nie można rzutować ani używać jako argument w operatorze instanceof,
-nie można używać tablic typów parametryzowanych (List<Integer>[] arrayOfLists = new List<Integer>[2]),
typ generyczny nie może rozszerzać (bezpośrednio lub pośrednio) Throwable,
nie można przeciążać metod, których argumenty sprowadzają się do tego samego typu.

```
public class Example {
public void print(Set<String> strSet) { }
public void print(Set<Integer> intSet) { }
}
```

- Napisał kod z użyciem wildcarda i zapytał czy lista.put(0, obj) się wykona i dlaczego nie
- Napisać metodę która korzysta z typu generycznego w klasie, która z niego nie korzysta

Wykład 5

- **Czym są strumienie I/O**

  An *I/O Stream* represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

  Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects. Some streams simply pass on data; others manipulate and transform the data in useful ways.

  No matter how they work internally, all streams present the same simple model to programs that use them: A stream is a sequence of data. A program uses an *input stream* to read data from a source, one item at a time, a program uses an *output stream* to write data to a destination, one item at time.

- **Opisz rodzaje strumieni**

  Strumienie bajtowe traktują dane jak zbiór ośmiobitowych bajtów.Wszystke strumienie bajtowe rozszerzają klasy `InputStream` (dane przychodzące do programu) lub `OutputStream` (dane wychodzące z programu).

  `CopyBytes` seems like a normal program, but it actually represents a kind of low-level I/O that you should avoid. Since contains character data, the best approach is to use character streams, as discussed in the next section. There are also streams for more complicated data types. Byte streams should only be used for the most primitive I/O.

  So why talk about byte streams? Because **all other stream types are built on byte streams.**

  Strumienie znakowe automatycznie konwertują dane tekstowe do formatu Unicode (stosowanego natywnie w Javie). Konwersja jest dokonywana w oparciu o ustawienia regionalne komputera, na którym uruchomiono JVM , lub jest sterowana “ręcznie” przez programistę. Strumienie znakowe rozszerzają klasy Reader (dane przychodzące do programu) lub Writer (dane wychodzące z programu) Strumienie znakowe wykorzystuję do komunikacji **strumienie bajtowe a same zajmują się konwersją danych**

  Strumienie znakowe buforowane umożliwiają odczytywanie tekstu linia
  po linii. Istnieją cztery klasy buforowanych strumieni: BufferedInputStream i
  BufferedOutputStream są strumieniami bajtowymi, podczas gdy
  BufferedReader i BufferedWriter odpowiadają za przesył znaków. Aby
  wymusić zapis danych poprzez wyjściowy, buforowany strumień,
  można użyć metody flush().
  It often makes sense *to write out a buffer at critical points, without waiting for it to fill. This is known as flushing the buffer*.Some buffered output classes support *autoflush*, specified by an optional constructor argument. When autoflush is enabled, certain key events cause the buffer to be flushed. For example, an autoflush `PrintWriter` object flushes the buffer on every invocation of `println` or `format`. See Formatting for more on these methods. To flush a stream manually, invoke its `flush` method. The `flush` method is valid on any output stream, but has no effect unless the stream is buffered.

  Strumienie binarne pozwalają efektywniej zarządzać zasobami. Istnieją dwa podstawowe rodzaje strumieni:
  -strumienie danych: DataInputStream i DataOutputStream:
  DataOutputStream dos = new DataOutputStream(System.out);
  dos.writeUTF("Grzegrz\u00f3\u0142ka");
  dos.writeInt(12345);
  dos.close();
  -strumienie obiektowe: ObjectInputStream i ObjectOutputStream:
  ObjectOutputStream oos = new ObjectOutputStream(System.out);

oos.writeObject("Grzegrz\u00f3\u0142ka");
oos.close();

- **Jak możemy pobierać dane użytkownika z klawiatury?**
  Scanner (java.util.Scanner) pozwala na przetwarzanie tokenów (domyślnie rozdzielonych przez Character.isWhitespace(char c)), obiekt należy zamknąć ze względu na strumień, z którym jest związany. Aby zmienić zachowanie obiektu Scanner, można skorzystać z metody: useDelimiter(). Przykładowo s.useDelimiter(",\\s*"); zmienia znak rozdzielający na przecinek po którym następuje dowolna liczba "białych spacji".

```
System.out.println("The square root of " + i + " is " + r + ".");
System.out.format("The square root of %d is %.4f \n", i, r );
System.out.format( Locale.US , "Pierwiastek z %.1f to %.4f\n", d, s);
System.out.printf( Locale.US , "Pierwiastek z %.1f to %.4f\n", d, s);
```

- **Wszystko o ResourceBundle(dokładne metody) odrazu opisać Properties i Locale**

  Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a string for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.
  This allows you to write programs that can:

  - be easily localized, or translated, into different languages
  - handle multiple locales at once
  - be easily modified later to support even more locales

```
public class LocalizationExample {
   public static void main(String[] args){
   ResourceBundle rb = ResourceBundle.getBundle("resources");
   for(String key: rb.keySet())
       System.out.println(key + ": " + rb.getString(key));
   }
}
......
Locale currentLocale;
        ResourceBundle messages;
        currentLocale = new Locale(language, country);
        messages = ResourceBundle.getBundle("MessagesBundle", currentLocale);
        System.out.println(messages.getString("greetings"));
        System.out.println(messages.getString("inquiry"));
        System.out.println(messages.getString("farewell"));.....
```

  Statyczna metoda getBundle("resources") jest równoważna wywołaniu `getBundle("resources", Locale.getDefault(), this.getClass().getClassLoader())`.
  i za pomocą bieżącego ClassLoadera poszukuje pliku o nazwie: `baseName +" "+ language +" "+ script +" "+ country +" "+ variant +".properties"`. Konkretna nazwa pliku jest ustalana na podstawie ustawien regionalnych systemu operacyjnego (Locale.getDefault()), np. resources_en_US_WINDOWS_VISTA.properties.

  Resource Bundle to mechanizm dostarczający zestaw zasobów (takich jak teksty, obrazy, pliki konfiguracyjne) w aplikacjach Javy. Pozwala na oddzielenie kodu aplikacji od danych, tak aby można było łatwo przetłumaczyć, zlokalizować i dostosować aplikację do różnych języków, krajów i ustawień regionalnych.
  Resource Bundle zawiera mapowanie klucza na wartość, gdzie klucz jest łańcuchem znaków służącym do identyfikacji wartości zasobów, a wartość może być ciągiem znaków, obrazem lub innym typem zasobu.
  W praktyce, pliki Resource Bundle są plikami tekstowymi, które zawierają pary klucz-wartość dla różnych języków. Aplikacja Javy może następnie pobrać odpowiedni zestaw zasobów dla danego języka i kraju z pliku Resource Bundle i wykorzystać go do wyświetlenia tekstu i innych zasobów w aplikacji.

- **Metody wieloargumentowe**

```
public static void multiint(int... ints){
  for (int i=0; i<ints.length; i++)
    System.out.println(ints[i]);
    System.out.println();
    for(int i: ints)
      System.out.println(i);
}
public static void main(String[] args){
  multiint(123,34,65,76,44,11,0);
  multiint();
  multiint(12, 28);
}
```

- **ClassLoader, Jak dostać instancje klasy z bytecodu (classloader metoda defineClass)**

  A class loader is an object that is responsible for loading classes. The class ClassLoader is an abstract class. Given the binary name of a class, a class loader should attempt to locate or generate data that constitutes a definition for the class. A typical strategy is to transform the name into a file name and then read a "class file" of that name from a file system. Every `Class` object contains a `reference` to the ClassLoader that defined it.

```
class NetworkClassLoader extends ClassLoader {
 String host;
 int port;
 public Class findClass(String name) {
     byte[] b = loadClassData(name);
     return defineClass(name, b, 0, b.length);
}
 private byte[] loadClassData(String name) {
 // wczytywanie bytecode'u klasy z określonej
 // lokalizacji sieciowej...
 }}}
```

- **Interfejs Serializable, co to serializacja, jakie metody się do tego stosuje.Jak potem zapisujemy klasę, która spełnia wymagania?,Opisac jak wyglada proces serializacji**

  Podstawowym zastosowaniem strumieni obiektowych jest serializacja. Klasa wspierająca serializację musi implementować interfejs Serializable. Jeśli obiekty tej klasy wymagają specjalnego traktowania podczas serializacji należy zaimplementować metody:
  `private void writeObject (java.io.ObjectOutputStream out)throws IOException;`
  `private void readObject( java.io.ObjectInputStream in)throws IOException, ClassNotFoundException;`
  Serializacja to mechanizm pozwalający przedstawić objekt jako **sekwencję bitów zawierających dane o obiekcie i informacje o typie obiektu i typie danych przechowywanych w obiekcie**, po serializacji i zapisaniu do pliku obiekt może być **deserializowany**, czyli odtworzony z wykorzystaniem serializacji. Proces ten jest niezależny od JVM, czyli ser. i deser. można wykonać na różnych platformach.

- **Strumienie(DataOutput,ObjectOutput)**

  Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain the methods for serializing and deserializing an object.

  The ObjectOutputStream class contains many write methods for writing various data types, but one method in particular stands out −
  `public final void writeObject(Object x) throws IOException`
  The above method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains the following method for deserializing an object
  `public final Object readObject() throws IOException, ClassNotFoundException`
  This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.

- **Strumienie kompresujące, Strumienie kompresujące ZIP**
  służą one do obsługi formatów gzip, zip, jar
  GZIPOutputStream nie zapisuje danych do pliku tylko je przetwarza (kompresuje). Do zapisu wykorzystuje on strumień, którego instancje dostaje w konstruktorze (tutaj FileOutputStream).
  Format ZIP obsługuje archiwa złożone z wielu plików, ZipEntry to znacznik informujący, że następujące po nim dane należą do wskazanego pliku

- **Jar**

  Java wyróżnia także szczególny rodzaj archiwum ZIP: JAR (JarOutputStream, JarInputStream). **Archiwa JAR zawierają pliki klas wraz z dodatkowymi zasobami potrzebnymi do działania aplikacji.** Podstawowe zalety dystrybucji programów w postaci plików jar to: bezpieczeństwo: archiwa mogą być cyfrowo podpisywane, kompresja: skrócenie czasu ładowania apletu lub aplikacji, zarządzanie

zawartością archiwów z poziomu języka Java, zarządzanie wersjami na poziomie pakietów oraz archiwów (Package Sealing, Package Versioning), przenośność. Archiwum jar tworzy sie używając komendy jar, np:

```
jar  cf  archiwum.jar klasa1.class klasa2.class ...
```

(c-tworzenie pliku, f-zapis do pliku, a nie na stdout, m-dołączenie pliku manifest,C-zmiana katalogu w trakcie działania archiwizatora)

W archiwum jar znajduje się katalog META-INF a w nim plik **MANIFEST.MF** zawierający dodatkowe informacje o archiwum. Przykładowa zawartość:

```
Manifest-Version: 1.0
Created-By: 1.5.0-b64 (Sun Microsystems Inc.)
Ant-Version: Apache Ant 1.6.5

Main-Class: pl.edu.uj.if.wyklady.java.Wyklad06
```

mówi, że po uruchomieniu archiwum wykonana zostanie metoda main(String[] args) zawarta w klasie Wyklad06 znajdującej się w pakiecie pl.edu.uj.if.wyklady.java. Uruchomienie pliku jar: `java -jar archiwum.jar`

- program taki ze pobiera dane z klawiatury i zapisuje do pliku
- Otworzyć plik, przeczytać zawartość i skompresować do gzipa
- Otworzyć plik"args[0]", skompresować GZipem do "args[0]+.gz" (Zamykać strumień!)
- **Napisać program który gzipem ztaruje plik tekstowy**

```
public class Kompresja{
    public static void main(String[] args) throws IOException {
        FileInputStream fis =null;
        try{
            fis = new FileInputStream(new File("tekst.txt"));
            FileOutputStream fos = new FileOutputStream("output.txt");
            ZipOutputStream zipOut = new ZipOutputStream(fos);
            ZipEntry zipEntry = new ZipEntry("tekst.txt");
            zipOut.putNextEntry(zipEntry);
            byte[] bufor = new byte[100];
            int length=0;
            while ((length = fis.read(bufor))>=0) {
                zipOut.write(bufor, 0, length);
            }
        }catch(Exception e){
            System.out.println("oopsie exception :(");
        }finally{
            if(fis!=null)
                fis.close();
        }
    }
}
```

- **Napisz klase, wpisz w nią int i arraylist, następnie ją zesrializuj,** podobne poniżej
- **Napisać serializable, które przekaże do pliku Vector oraz int**

```
import java.io.*;
public class SerializeDemo {

   public static void main(String [] args) {
      Object obj = new Object();
      obj.vector = someVector;
      obj.int = someInt;

      try {
         FileOutputStream fileOut =new FileOutputStream("/tmp/employee.ser");
         ObjectOutputStream out = new ObjectOutputStream(fileOut);
         out.writeObject(obj);
         out.close();
         fileOut.close();
         System.out.printf("Serialized data is saved in /tmp/employee.ser");
      } catch (IOException i) {
         i.printStackTrace();
      }
   }
}
//DESERIALIZACJA
public static void main(String [] args) {
      Employee e = null;
      try {
         FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
         ObjectInputStream in = new ObjectInputStream(fileIn);
         e = (Employee) in.readObject();
         in.close();
         fileIn.close();
```

```
      } catch (IOException i) {
         i.printStackTrace();
         return;
      } catch (ClassNotFoundException c) {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }
   }
}
```

- **Napisać program który jako pierwszy argument wywołania programu podaje nazwę pliku a następne argumenty to dane które chcemy zapisać do tego pliku**

```
public class DoPliku {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.err.println("Usage: java DoPliku <file_name> <data...>");
            System.exit(1);
        }
        String fileName = args[0];
        try {
            FileOutputStream outputStream = new FileOutputStream(fileName);
            for (int i = 1; i < args.length; i++) {
                String data = args[i];
                outputStream.write(data.getBytes());
            }
            outputStream.close();
            System.out.println("Data saved to file: " + fileName);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- **Napisz program, który kopiuje tekst z jednego pliku do drugiego**

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

Wykład 6

- **Procesy, jak uruchomić,tworzenie procesów,co zwraca Runtime.getRuntime().exec, co pobiera w argumencie.**
  Zwraca objekt Process do zarządzania nowym podprocesem, a przyjmuje komendę odpowiednią dla danego so np. w linuxie “ls -s”

  The `ProcessBuilder.start()` and `Runtime.exec` methods create a native process and return an instance of a subclass of Process that can be used to control the process and obtain information about it. The class Process provides methods for performing input from the process, performing output to the process, waiting for the process to complete, checking the exit status of the process, and destroying (killing) the process. The methods that create processes may not work well for special processes on certain native platforms, such as native windowing processes, daemon processes, Win16/DOS processes on Microsoft Windows, or shell scripts. By default, the created

subprocess does not have its own terminal or console. All its standard I/O (i.e. stdin, stdout, stderr) operations will be redirected to the parent process, where they can be accessed via the streams obtained using the methods `getOutputStream()`, `getInputStream()`, and `getErrorStream()`. The parent process uses these streams to feed input to and get output from the subprocess. Because some native platforms only provide limited buffer size for standard input and output streams, failure to promptly write the input stream or read the output stream of the subprocess may cause the subprocess to block, or even deadlock.
Where desired, subprocess I/O can also be redirected using methods of the `ProcessBuilder` class.
The subprocess is not killed when there are no more references to the `Process` object, but rather the subprocess continues executing asynchronously.
There is no requirement that a process represented by a `Process` object execute asynchronously or concurrently with respect to the Java process that owns t

- Czy jest możliwe by utworzony w javie proces czekał? Jeśli tak to jak to zrobić.
  Jak stworzyć proces w javie.
- **kod, jak wywołać nowy wątek nie rozszerzając klasy i nie implementując żadnego interfejsu**

```
Thread t = new Thread( ()->{instrukcje, które by były w run()....} );
```

- **synchronized w wątkach, co to, jak uzywac**

  Wątki mogą się komunikować przez dowolne współużytkowane zasoby (np. referencje do obiektów). Jest to bardzo efektywne, jednak może powodować problemy, gdy kilka wątków korzysta z jednego zasobu. Oznaczenie metod słowem `synchronized` powoduje, że w danej chwili może być wykonywana tylko jedna z nich (i tylko przez jeden wątek) np. `public synchronized void increment(){…}`
  Każdy obiekt posiada powiązaną z nim blokadę wewnętrzna (intrinsic lock). Jeśli wątek chce uzyskać wyłączny dostęp do obiektu lub jego atrybutów może skorzystać z tej blokady. Wątek musi jednoznacznie wskazywać obiekt, z którym jest związana używana przez niego blokada. Taki tyb blokad nazywamy **synchronizacją na poziomie instrukcji** (synchronized statements)

  ```
  public void addName(String s) {
      synchronized(this) {
      name = s;
      counter++;
      }
      nameList.add(name);
  }
  ```

- **Co to są blokady (Lock) i jak synchronizować nimi wątki**
  Blokada drobnoziarnista np. jeśli mamy dwa liczniki i chcemy je zabezpieczyć, ale nie chcemy blokować obu na raz to robimy `synchronized(lock1)` w jednej metodzie, a w drugiej `synchronized(lock2)`
- **Volatile - czym jest, gdzie się używa, co robi+ czy można używać poza wątkiem**

  Typowo dostęp do zmienne/referencji nie jest realizowany jako pojedyncza operacja. Aby takie operacje (odczytu/zapisu zmiennej) były **atomowe** należy oznaczyć ją słowem volatile

  The purpose of volatile is to ensure that when one thread writes a value to the field, the value written is "immediately available" to any thread that subsequently reads it. Conversely,other threads reading the value are "forced" to take that immediately available value, rather than relying on a value previously read and cached by those other threads. Without volatile or any other form of synchronization, then modern CPU architecture and optimisations can lead to unexpected results. A thread can work on a cached, out-of-date copy of the value in question, or not write it back to main memory in time for it to be accessed by another thread as expected. Under the hood, volatile causes reads and writes to be accompanied by a special CPU instruction known as a memory barrier. The memory barrier forces the required execution ordering and memory visibility between the different CPU cores, so that to the programmer, reading and writing to the shared volatile field then behaves as expected. You can think of volatile informally as being equivalent to automatically putting synchronized around each individual read or write of the field in question. And synchronized can be used to achieve a similar effect (also involving memory barriers under the hood).But synchronized is a more "heavyweight" solution under the hood because it also entails lock management, whereas volatile is lock-free
- **Typowe problemy współbieżności**

  zakleszczenia (deadlock) – wątki blokują wzajemnie zasoby potrzebne im do dalszego działania (pięciu filozofów).
  zagłodzenia (starvation) - jeden wątek przez cały blokuje zasób potrzebny innym wątkom.
  livelock – “odwrotność” deadlocka – wątek reaguje na zachowanie drugiego wątku, które jest reakcją na zachowanie pierwszego wątku.

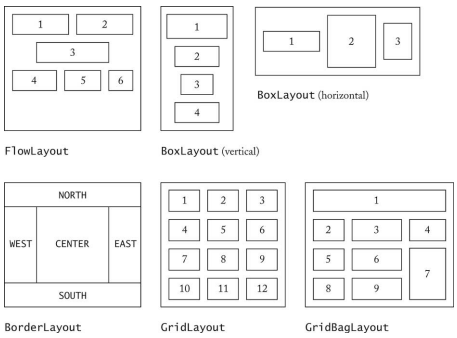
- Wątki, jak działa wait() i t.join()
- **Co robi join**, później dopytał czym się to różni od tego jakbym nnormalnie zakończył wątek returnem jak nie wiedziałem to kazał czytać dokumentację xdd

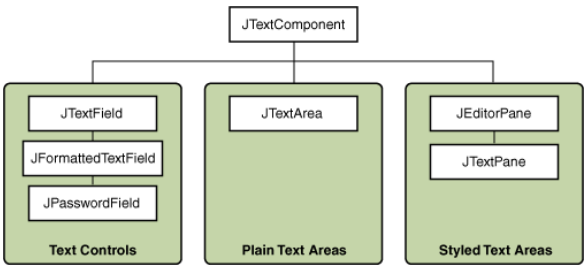
  The `join` method allows one thread to wait for the completion of another. If `t` is a `Thread` object whose thread is currently executing, `t.join();` causes the current thread to pause execution until `t`'s thread terminates. Overloads of `join` allow the programmer to specify a waiting period. However, as with `sleep`, `join` is dependent on the OS for timing, so you should not assume that `join` will wait exactly as long as you specify. Like `sleep`, `join` responds to an interrupt by exiting with an `InterruptedException`.
- **wait i notify, czy można zawsze wywołać (nie)**
  wait() uwalnia blokadę po czym czeka i znowu włącza blokadę przed wykonaniem jakiś działań, czyli mówi wątkowi że ma odblokować lock i iść spać aż zostanie powiadomiony notify()
  notify() budzi jeden wątek, który wywołał wcześniej wait(), nie spuszcza ona blokady na zasobie tylko mówi wątkowi, że ten może się już obudzić, ale zasób jest odblokowany w momencie, w którym zsynchronizowany blok się w całości wykona
  notifyAll() budzi wszystkie wątki, które wywołały wait() na danym obiekcie, zwykle jako pierwszy zacznie wtedy działać wątek o najwyższym priorytecie
  In general, a thread that uses the `wait()` method confirms that a condition does not exist (typically by checking a variable) and then calls the `wait()` method. When another thread establishes the condition (typically by setting the same variable), it calls the `notify()` method. The wait-and-notify mechanism does not specify what the specific condition/ variable value is. It is on developer's hand to specify the condition to be checked before calling `wait()` or `notify()`.
- **ExecutorService na czym polega,co zwraca metoda submit**
  Executory to obiekty służące do zarządzania wątkami, oddzielają wątek od zarządzania nim, Executor jest rozszerzany przez ExecutorService, który dodatkowo umożliwia uruchamianie wątków Callable(jak Runnable, ale mogą coś zwracać)
  ThreadPools to są pule wątków, zbiory wspólnie zarządzanych wątków
  `ExecutorService es = Executors.newFixedThreadPool(16)`
  `es.submit(new Callable<Object>{override metody call zwracającej Obj});`
  submit() submits a Callable or a Runnable task to an ExecutorService and returns a result of *type Future*
  `executorService.shutdown();`
- **ForkJoinPool co trzeba żeby swój algorytm na tym odpalić**
  Jest to rodzaj puli wątków korzystający z algorytmu dziel i zwyciężaj, w którym dzielimy pracę na mniejsze podzadania, kązdy z podziałów nazywamy forkiem,
  musimy mieć klasę, która jest podklasą forkJoinTask i w niej opisać swój algorytm i potem na obiekcie tej klasy wywołać za pomocą invoke
  `pool.invoke(myForkJoinTask)`
- Napisz program, który stworzy 5 wątków, każdy z nich wypisze inną literę
- blokada wewnętrzna oraz drobnoziarnista. Pokazał kod, gdzie uruchamiał 4 wątki na tą samą metodę i trzeba było podać sposób, by były odpowiednio zsynchronizowane niezależnie.

Wykład 7 SWING

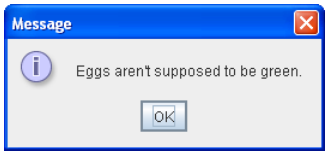
- **Layout manager - co to, rodzaje, czy jframe ma defaultowego managera**
  Dla JFrame default to BorderLayout, dla JPanel to FlowLayout
  FlowLayout: It arranges the components in a container like the words on a page. It fills the top line from left to right and top to bottom. The components are arranged in the order as they are added i.e. first components appears at top left, if the container is not wide enough to display all the components, it is wrapped around the line. Vertical and horizontal gap between components can be controlled. The components can be left, center or right aligned.
  BorderLayout: It arranges all the components along the edges or the middle of the container i.e. top, bottom, right and left edges of the area. The components added to the top or bottom gets its preferred height, but its width will be the width of the container and also the components added to the left or right gets its preferred width, but its height will be the remaining height of the container. The components added to the center gets neither its preferred height or width. It covers the remaining area of the container.
  GridLayout: It arranges all the components in a grid of equally sized cells, adding them from the left to right and top to bottom. Only one component can be placed in a cell and each region of the grid will have the same size. When the container is resized, all cells are automatically resized. The order of placing the components in a cell is determined as they were added.
  GridBagLayout: It is a powerful layout which arranges all the components in a grid of cells and maintains the aspect ration of the object whenever the container is resized. In this layout, cells may be different in size. It assigns a consistent horizontal and vertical gap among components. It allows us to specify a default alignment for components within the columns or rows.
  BoxLayout: It arranges multiple components in either vertically or horizontally, but not both. The components are arranged from left to

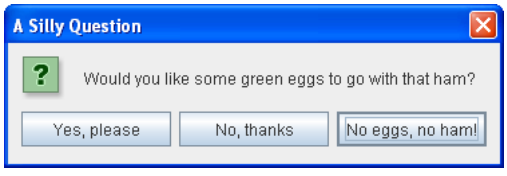
right or top to bottom. If the components are aligned horizontally, the height of all components will be the same and equal to the largest sized components. If the components are aligned vertically, the width of all components will be the same and equal to the largest width components.
CardLayout: It arranges two or more components having the same size. The components are arranged in a deck, where all the cards of the same size and the only top card are visible at any time. The first component added in the container will be kept at the top of the deck. The default gap at the left, right, top and bottom edges are zero and the card components are displayed either horizontally or vertically.





- **czym się różni JEditorPane od JTextArea**
  JEditorPane pozwala na wiele różnych czcionek, personalizację, można dodać też embedded obrazy czy komponenty, można stworzyć z doc StyledDocument w konstr.
  JTextArea pozwala na wiele lin edytowalnego tekstu, ale cały musi być w jednej czcionce
- **pack()**
  dopasowanie rozmiarow okna do umieszczonych w nim komponentow
- **JFileChooser (jak stworzyć, co dostaje sie na wyniku, jak dostać sie do plików które wybrał użytkownik, czy można wybrać więcej niż jeden, co w JPanelu zmienia podanie go jako argument w showOpenDialog, czy można null)**
  FileChoosery zapewniają GUI możliwość nawigowania po systemie plików, można wtedy wybrać katalog, plik z listy lub wpisując jego nazwę. Aby to zrobić dodajemy do odpowiedniego kontenera JFileChooser i do odczytywania/zapisywania w plikach korzystamy z strumieni io.
  Można wybrać wiele plików i zwrócić tablicę plików zamiast jednego obiektu File
  `showOpenDialog()` zwraca `JFileChooser.APROVE_OPTION` lub `JFileChooser.CANCEL_OPTION`, jako argument przyjmuje komponent określający gdzie wyświetlić okno dialogowe, jeśli podamy null jako argument to okno dialogowe będzie wyświetlane w obrębie aplikacji, która wywołała funkcję showOpenDialog().
- **JOptionPane** (powiedzialem ze sa tam 4 metody przykladowe np showMessageDialog, showInputDialog i potem na kartce mi kazal napisac showInoutDialog)
  showMessageDialog displays a modal dialog with one button, which is labeled "OK" (or the localized equivalent). You can easily specify the message, icon, and title that the dialog displays. Mamy opcje: `default, WARNING_MESSAGE, ERROR_MESSAGE, PLAIN_MESSAGE, INFORMATION_MESSAGE`
  showOptionDialog działa jak wyżej tylko zamiast okejki możemy dodać więcej przycisków
  showInputDialog() zwraca nam objekt, zazwyczaj Stringa z odpowiedzią użytkownika
  showInternalMessageDialog ??





- **Jak dodawać scrolla**

```
// pole tekstowe webpage umieszczamy wewnatrz panela
// scrollowanego. Dzieki temu zawartosc okienka bedzie mogla
// zajmowac wiecej miejsca niz widok
JScrollPane sp = new JScrollPane(this.webpage);
```

```
// zakladka “page” bedzie zawierac webpage (wewnatrz JScrollPane)
tp.add("page", sp);
// zakladka“html”bedzie zawierac htmlPage (wewnatrz JscrollPane)
sp = new JScrollPane(this.htmlPage);
tp.add("html", sp);
```

- **Jfilechooser wybranie pliku i wypisanie do niego hello**
- **Napisz JFileChooser, który wyświetli zawartość pliku, który wybierze użytkownik**

```
public class Display extends JPanel implements ActionListener{
  private JFileChooser filechooser;
  private......

  public Display(){
    ...
  }
  public void actionPerformed(ActionEvent e){
        if(e.getSource() == button){
            returnValue = filechooser.showOpenDialog(null);
            if(returnValue == JFileChooser.APPROVE_OPTION){
                file = filechooser.getSelectedFile();
                try{
                    br = new BufferedReader(new FileReader(file));
                    bw = new BufferedWriter(new FileWriter(file));
                    bw.flush();
                    bw.write("hello");//wpisanie hello
                    bw.flush();
                    while((currentLine = br.readLine()) != null){
                        System.out.println(currentLine);//wypisanie zawartości
                    }
                } catch (Exception error){
                    error.printStackTrace();
}}}}
public static void main(String args[]){
        JFrame frame = new JFrame("My GUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(new Display());
        frame.pack();
        frame.setVisible(true);
}
```

- Kod w swingu , 3 pola tekstowe pod sobą, 3 Jbuttony pod sobą
- Napisać pole do wpisywania tekstu i pod nim przycisk. Gdy klikniemy w przycisk to wypisuje w konsoli ten tekst.
- Napisać program, który czyta tekst ze standardowego wejścia i wyświetla go w polu tekstowym w nowym okienku (czyli Swing).
- Napisać aplikacje okienkową z dwoma JButtonami które reagują na kliknięcia
- **Napisać w jednej linijce JOptionPane, który wyświetli Hello i będzie miał przycisk OK**
- **Napisać okienko z przyciskiem, który wyświetla okienki z napisem Hello i przyciskiem ok (JOptionPane)**

```
JOptionPane.showMessageDialog(frame,"Hello");
```

Wykład 8

- **URL(jak uzyskać, jakie protokoły obsługuje), URLConnection(jak się tworzy, do czego służy, czy może służyć do przesyłania danych) różnice i jak odbierac dane z serwera**
  URL to podstawowa klasa identyfikująca zasoby w internecie, uzyskujemy przez `URL url = new URL(”http://www.google.pl/”)`
  URLConnection umożliwia nawiązanie połączenia z zasobem reprezentowanym przez URL `URLConnection con=url.openConnection()`
  i odczyt/zapis danych do wskazanego zasobu przez BufferedReader lub Writer. Obsługiwane protokoły to: http, https, ftp, file, jar oraz inne jeśli zaimplementujemy `URLStreamHandler`
- **Socket i ServerSocket jak działa, w jaki sposób nawiązują połączenie co mają w środku, w jaki sposób przekazują dan, co je używa, co przyjmują, .connect(), co się dzieje jak nic się nie połączy i close()**

  Socket – klasa reprezentująca gniazdo służące do nawiązywania
  połączenia, wysyłania I odbierania danych,
  ServerSocket – klasa reprezentująca gniazdo oczekujące na
  przychodzące żądania połączeń

- **Co robi accept w Socket**
  w socket nie ma accept, a accept w serversocket akcpetuje prośbę o połączenie od socket
- **Napisać server, który wysyła hello do klienta**

```
public class Server{
  public static void main(String[] args){
    ServerSocket ss = new ServerSocket(nrportu);
    Socket s = ss.accept()
    PrintWriter pw = new PrintWriter(s.getOutputStream);
    pw.println("hello");
    pw.flush();
  }
}
```

- **SSLServerSocket**

  Protokół SSL umożliwia bezpieczną (szyfrowaną) transmisję danych poprzez niezabezpieczoną sieć. Dodatkowo SSL umożliwia autoryzację stron komunikacji. W tym celu wykorzystywany jest mechanizm certyfikatów. Za transmisję z użyciem protokołu SSL odpowiedzialne są klasy zgrupowane w pakiecie javax.net.SSL

- **Napisać klienta ssl (i przy okazji serwer xd)**

```
public class SSLClient{
  public static void main(String[] args){
    SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
    SSLSocket s = (SSLSocket)factory.createSocket(hostname,port);
    /*i dalej jazda robimy co chcemy*/
  }
}
public class SSLServer{
  public static void main(String[] args){
    SSLServerSocketFactory factory = (SSLServerSocketFactory)SSLServerSocketFactory.getDefault();
    SSLServerSocket ss = (SSLServerSocket)factory.createServerSocket(port);
    SSLSocket s = (SSLSocket)ss.accept();
    /*i dalej jazda robimy co chcemy*/
  }
}
```

- **generowanie klucza, jakie argumenty się podaje w komendzie keytool, jak użyć certyfikatu w serwerze i kliencie**

  wygenerowanie klucza: `keytool -genkey -keystore mySrvKeystore -keyalg RSA`
  -genkey oznacza, że zostaną wygenerowane pary kluczy publiczny-prywatny, -keystore i kolejny argument to miejsce gdzie przechowamy wygenerowane dane, -keyalg to algorytm wykorzystany do stworzenia kluczy
  uruchomienie serwera: `java -Djavax.net.ssl.keyStore=mySrvKeystore -Djavax.net.ssl.keyStorePassword=123456 EchoServer`
  uruchomienie klienta: `java -Djavax.net.ssl.trustStore=mySrvKeystore -Djavax.net.ssl.trustStorePassword=123456 EchoClient`
  dodatkowe parametry wywołania pozwolą zobaczyć informacje
  związane z połączeniem SSL:
  `-Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol`
  `-Djavax.net.debug=ssl`

  Domyślnie tylko jedna strona komunikacji (serwer) musi potwierdzać swoją tożsamość. Aby wymusić autoryzację klienta należy użyć metod: `setNeedClientAuth(true)` lub `setWantClientAuth(true)` wywołanych na rzecz obiektu SSLServerSocket.
  Jeśli chcemy aby żadna ze stron nie musiała potwierdzać swojej tożsamości musimy zmienić domyślne algorytmy kodowania. Najłatwiej zrobić to tworząc własne rozszerzenie klasy
  SSLSocketFactory.

- **Echo serwer(dostaje dane od klienta i je zwraca)**

  W serwerze nie ma problemu związanego z transmisją strumieniową ponieważ dane są przetwarzane bajt po bajcie, w związku z czym nie sytuacja, gdy dane dotrą w różnych pakietach nie spowoduje żadnych efektów ubocznych

```
import java.io.;
import java.net.;
public class ServerExample {
  // gniazdo oczekujace na polaczenia
  private static ServerSocket ss;
  public static void main(String[] args) throws IOException{
    // tworzymy gniazdo,ktore oczekuje na przychodzace polaczenia
```

```
    // na porcie przekazanym jako argument wywolania programu
    ss = new ServerSocket(Integer.valueOf(args[0]));//port
    // nieskonczona petla
    while(true){
    // akceptujemy polaczenie,dostajemy gniazdo do komunikacji
    // z klientem
    Socket s = ss.accept();
    // strumienie
    InputStream is = s.getInputStream();
    OutputStream os = s.getOutputStream();
    int b;
    // czytamy, piszemy na konsoli i odsylamy
    while((b=is.read())!=-1){
      System.out.print((char)b);
      os.write(b);
      }
    s.close();
    }
  }
}
```

- **Napisać serwer TCP**

```
import java.io.IOException;/*ze strony Oramusa*/
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

public class SimpleServer {
  public static void main(String[] args) {
    final int PORT = 55555;
    try {
      ServerSocket so = new ServerSocket( PORT ); // tworzymy gniazdo oczekujace
      while ( true ) {
        System.out.println( "A teraz sobie poczekamy" );
        Socket s = so.accept(); // to blokujaca metoda, czekamy na klienta
        System.out.println( " - o jest klient " );
        PrintWriter out = new PrintWriter( s.getOutputStream() );
        out.println( "Witaj kimkolwiek jestes !" );
        out.flush();
        try {
          Thread.sleep( 5000 ); // spimy przez 5 sekund
        }
        catch ( InterruptedException e ) {}
        out.println( "Witaj jeszcze raz kimkolwiek jestes !" );
        out.flush();
        out.close();
    }}
    catch ( IOException ex ) { ex.printStackTrace(); }
  }}
```

- serwer ssh(nie ogarniam czym się różni od innych)
- **WebStart**
  opisać JWS oraz JNLP. W jakim formacie są pliki JNLP, co się w nich znajduje i jak działają

  Technologia Java Web Start jest stosowana do lokalnego uruchamiania programów w Javie umieszczonych w sieci. JWS:
  ● jest w pełni niezależna od używanych przeglądarek internetowych,
  ● umożliwia automatyczne pobranie właściwej wersji środowiska JRE,
  ● pobierane są tylko pliki, które zostały zmienione,
  ● obsługuje prawa dostępu do zasobów lokalnego komputera (dysk, sieć, itp.),
  ● do opisu zadania do uruchomienia wykorzystuje pliki jnlp (Java Network Launch Protocol), które umieszczane są na serwerze www

Wyklad 9 XML

- **budowa pliku XML**

  Plik XML (Extensible Markup Language) w Javie jest to plik tekstowy, który jest często używany do przechowywania danych lub informacji w strukturalnej formie. Plik XML składa się z tagów, które są umieszczone wewnątrz nawiasów ostrych "<>" oraz wartości, które znajdują się między nimi. Tagi w pliku XML reprezentują różne elementy i atrybuty, a wartości są informacjami z nimi związanymi. W Javie, pliki XML mogą być używane do przechowywania konfiguracji aplikacji, danych o produktach lub usługach, danych o klientach, itp. Pliki XML mogą być odczytywane i przetwarzane przez różne aplikacje Javy za pomocą specjalnych bibliotek, takich jak DOM (Document Object Model) lub SAX (Simple API for XML). W Javie istnieją również specjalne biblioteki do

generowania plików XML, takie jak JAXB (Java Architecture for XML Binding), które umożliwiają tworzenie plików XML z obiektów Javy lub danych.

- **JAXB, szczegółowo, co to jak użyć itd**

  JAXB (Java Architecture for XML Binding) to standard serializacji XML dla obiektów Javy. Został on zintegrowany z JDK/JRE od wersji 1.6

- **JAXB, marshaller, metody, inicjalizacja kontekstu**
  W SAX, marshaller działa na zasadzie odwrotnej do parsera XML. Parser XML służy do konwersji danych XML na obiekty w języku Java, podczas gdy marshaller wykonuje operację odwrotną - konwertując obiekty w języku Java na dane XML. Marshallery są bardzo przydatne w przypadku, gdy konieczne jest przesyłanie danych z aplikacji w języku Java do innego systemu, który używa formatu XML jako standardu komunikacji.

```
public class JAXBExample{
 public static void main(String[] args) throws JAXBException {
     Person p = new Person("Hanka",33);
     File f = new File("person.xml");
     JAXBContext ctx = JAXBContext.newInstance(Person.class);
//"marshaller" to procesor, który jest odpowiedzialny za konwersję danych z formatu Java na format XML.
     Marshaller marshaller = ctx.createMarshaller();
     marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
     marshaller.marshal(p, System.out);
     marshaller.marshal(p, f);
     p = null;
     Unmarshaller unmarshaller = ctx.createUnmarshaller();
     p = (Person)unmarshaller.unmarshal(f);
     System.out.println(p);
 }
}
```

- **Co to jest ANT**
  Ant jest narzędziem umożliwiającym automatyzację procesów związanych z budowaniem programów. Jego podstawowe cechy to:
  ● konfiguracja zadań zapisana w formacie XML,
  ● wieloplatformowość – m. in. Linux, Unix (np. Solaris and HP-UX), Windows 9x i NT, OS/2 Warp, Novell Netware 6 oraz MacOS X.
  ● rozszerzalność w oparciu o klasy napisane w Javie

```
//Przykładowy plik konfiguracyjny dla anta (domyślnie build.xml)
<project name="project" default="default" basedir=".">
 <description>
 jakis opis
 </description>
 <target name="default" depends="depends" description="description">
 <jar destfile="bendmetter.jar" includes="**/*.class"
 excludes="tests/*.*" basedir="bin"></jar>
 </target>
 <target name="depends">
 </target>
</project>  //Aby go “wykonać” wpisujemy w konsoli polecenie ant
```

- **Wypytywał o sax i dom**
- **DOM,jak pobrać elementy, co zwraca getElementsByTagName**
  Parsery DOM (Document Object Model) tworzą drzewo reprezentującedane zawarte w dokumencie XML. Po zbudowaniu DOM przetwarzanie odbywa się na modelu w pamięci operacyjnej. Dobry do operacji na małych plikach i modyfikowania struktury XML. Przedstawia zawartość pliku XML jako węzły drzewa
- **SAX Parser- co to, opisać po kolei parsowanie + czym jest DefaultHadler i jak się z niego korzysta**
  SAX (Simple API for XML) w miarę czytania dokumentu wywołuje zdarzenia związane z parsowaniem. Parsery SAX są szybsze i nie wymagają tak dużej ilości pamięci jak DOM. Nie tworzy nowej reprezentacji jak DOM(drzewa) więc szybszy i lepszy dla dużych plików.
- **XmlEncoder i XmlDecoder**

```
//Inna metoda serializacji niektórych obiektów do plików tekstowychw formacie XML:
//XMLEncoder:
XMLEncoder e = new XMLEncoder(new FileOutputStream("jbutton.xml"));
e.writeObject(new JButton("Hello world"));
e.close();
//XMLDecoder:
XMLDecoder d = new XMLDecoder(new FileInputStream("jbutton.xml"));
```

```
obj = d.readObject();
d.close();
```

- **Co to defaulthandler, w jaki sposób obsłużyć konkretną rzecz z XML za pomocą defaulthandlera**
  `DefaultHandler` implementuje metody, które są wywoływane podczas przetwarzania dokumentu XML przez SAX. Jednym z zastosowań jest tworzenie własnych parserów SAX. Programista może stworzyć klasę dziedziczącą po `DefaultHandler` i zaimplementować w niej metody, które obsługują odpowiednie zdarzenia XML. Następnie tę klasę można użyć jako parsera SAX, który przetwarza dokument XML zgodnie z zaimplementowaną logiką.
  Te metody to m.in.:

```
void characters(char[] ch, int start, int length) – wywoływane przy odczycie znaku wewnątrz elementu,
void endDocument() – wywoływane gdy koniec dokumentu
void endElement(String uri, String localName, String qName) – koniec elementu,
void error(SAXParseException e) – błąd (możliwy do naprawienia),
void fatalError(SAXParseException e) – błąd,
void ignorableWhitespace(char[] ch, int start, int length) – ignorowany pusty znak,
void startDocument() – początek dokumentu,
void startElement(String uri, String localName, String qName,Attributes attributes) – początek elementu
void warning(SAXParseException e) – ostrzeżenie parsera
```

- **Napisać kod na kartce jak się parsuje saxem**

```
import java.io.IOException;
import java.net.URL;
import javax.xml.parsers.*;
import org.xml.sax.*;
public class SAXExample {
 public static void main(String[] args) throws SAXException,IOException, ParserConfigurationException{
   URL url = new URL("http://www.uj.edu.pl/...");
   SAXParserFactory f = SAXParserFactory.newInstance();
   SAXParser saxParser = f.newSAXParser();
   DefaultHandler handler = new ExampleSAXHandler();
   saxParser.parse(url.openStream(), handler);
 }
}
```

- **Napisać Parser DOM + jak dostać się do roota + jak dostać się do innych elementów**

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class XMLParserDOM {
  public static void main(String[] args) {
    try {
      File xmlFile = new File("nazwa_pliku.xml");
      DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse(xmlFile);// tworzenie modelu DOM na podstawie zrodla XML (parsowanie XML'a)
      //doc.getDocumentElement().normalize();
      Element rootElement = doc.getDocumentElement();
      System.out.println("Root element: " + rootElement.getNodeName());
      NodeList nodeList = rootElement.getElementsByTagName("nazwa_tagu");
      for (int i = 0; i < nodeList.getLength(); i++) {
        Element element = (Element) nodeList.item(i);
        System.out.println("Element name: " + element.getNodeName());
        String attributeValue = element.getAttribute("nazwa_atrybutu");
        System.out.println("Attribute value: " + attributeValue);
        String elementValue = element.getTextContent();
        System.out.println("Element value: " + elementValue);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

- Zapis dokumentu w DOM

```
// domyslna fabryka transformatorow
TransformerFactory transformerFactory = TransformerFactory.newInstance();
// nowy tranformator
Transformer transformer = transformerFactory.newTransformer();
// wejscie transformatora (skad transformator bierze dane)
DOMSource source = new DOMSource(doc);
// wyjscie transformatora (gdzie “zmienione” dane zostana zapisane)
StreamResult result = new StreamResult(new File("file.xml"));
// uruchomienie transformatora – zapis DOM do pliku w formacie XML
transformer.transform(source, result);
```

Wyklad 10 SQL

- **JDBC, omówić sposoby połączenia z bazą danych}**
  Java Database Connectivity (JDBC) to specyfikacja określająca zbiór klas i interfejsów napisanych w Javie, które mogą być wykorzystane przez programistów tworzących oprogramowanie korzystające z baz danych. Implementacja JDBC jest dostarczany przez producentów baz danych.
  Jedną z ważniejszych zalet takiego rozwiązania jest ukrycie przed programistą kwestii technicznych dotyczących komunikacji z bazą danych. Dzięki temu ten sam program napisany w Javie może współpracować z różnymi systemami baz danych (np. Oracle, Sybase, IBM DB2). Wystarczy podmienić odpowiednie biblioteki implementujące JDBC.
  Korzystanie z interfejsu JDBC umożliwia jednolity sposób dostępu do różnych systemów bazodanowych. Jako przykład przedstawiono bazę HSQLDB. Jej ważną zaletą jest możliwość działania w ramach jednej Wirtualnej Maszyny Javy wraz z programem klienckim.

- **Nawiązanie połączenia z bazą danych i konkretnie jak to zrobic za pomoca drivermanagera**
  Istnieją dwie metody nawiązania połączenia z bazą danych:
  ● za pomocą klasy DriverManager:
  `String url = " jdbc:odbc: bazadanych";`
  `Connection con = DriverManager.getConnection(url, "login", "haslo");`
  ● za pomocą klasy DataSource i usługi JNDI:
  `Context ctx = new InitialContext();`
  `DataSource ds = (DataSource) ctx.lookup("jdbc/MojaDB") ;`
  `Connection con = ds.getConnection("myLogin", "myPassword") ;`
  DataSource reprezentuje źródło danych. Zawiera informacje identyfikujące i opisujące dane. Obiekt DataSource współpracuje z technologią Java Naming and Directory Interface (JNDI), jest tworzony i zarządzany niezależnie od używającej go aplikacji.
  Korzystanie ze źródła danych zarejestrowanego w JNDI zapewnia:
  ● brak bezpośredniego odwołania do sterownika przez aplikację,
  ● umożliwia implementację grupowania połączeń (pooling) oraz rozproszonych transakcji.
  Te cechy sprawiają, że korzystanie z klasy DataSource jest zalecaną metodą tworzenia połączenia z bazą danych, szczególnie w przypadku dużych aplikacji rozproszonych.

- **Jak w bazie danych otrzymać dany wynik, rezultat? (ResultSet)**
  Wyniki zwrócone w wyniku wykonania zapytania są dostępne poprzez obiekt typu ResultSet.

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM table");
while (rs.next()) {// odebranie i wypisanie wyników w bieżącym rekordzie
 int i = rs.getInt("a"); //nazwa kolumny
 String s = rs.getString("b");
 float f = rs.getFloat("c");
 System.out.println("Rekord = " + i + " " + s + " " + f);}
```

- **proszę opisać interfejs Statement" + dlaczego statement to interfejs, a nie klasa abstrakcyjna co to statement (to interfejs a nie klasa), skąd go pobieramy, jak go używać (np. executeQuery)**
  Interfajs Statement zapewnia metody służące do wykonywania zapytań do bazy danych, pozwala na otrzymanie wyników w postaci ResultSet. W ramach jednego obiektu Statement można wykonać sekwencyjnie kilka zapytań. Po zakończeniu używania obiektu zaleca się wywołanie metody `close()`.
  Czemu interfejs? Because there's *nothing* which can be provided as default implementation which will work with **any** database engine the world is aware of.
  `public ResultSet executeQuery (String sql)` is used to execute SELECT query. It returns the object of ResultSet.
  `public int executeUpdate (String sql)` is used to execute specified query, it may be create, drop, insert, update, delete etc.
  `public boolean execute (String sql)` is used to execute queries that may return multiple results
  `public int[] executeBatch ()` is used to execute batch of command

- **Co to jest HSQLD? Czym się różni od innych? Jaki tryb jest użyteczny (chodziło oczywiście o Stand Alone)?**
  HSQLDB to system zarządzania relacyjnymi bazami danych w całości napisany w Javie (open source). Niektóre własności:
  ● obsługa SQL'a,
  ● obsługa transakcji (COMMIT, ROLLBACK, SAVEPOINT), zdalnych procedur i funkcji (mogą być pisane w Javie), triggerów itp.
  ● możliwość dołączania do programów i appletów. Działanie w trybie read-only,
  ● rozmiar tekstowych i binarnych danych ograniczony przez rozmiar pamięci.
  tryby: Server(preferowany), WebServer(jeśli serwer może korzystać tylko z protokołu http, https), Servlet(ten sam protokół co webserver ale wymaga konteneru serwletów np.tomcat), Stand-Alone, wszystkie tryby pracy umożliwiają korzystanie z JDBC.

- **Czym jest tryb stand-alone w bazach HSQLDB?**
  W trybie stand-alone „serwer” bazy danych działa w ramach tej samej wirtualnej maszyny Javy, co korzystający z niego program „kliencki”. Przykład uruchomienia bazy:
  `Connection c+DriverManager.getConnection("jdbc:hsqldb:file:/opt/db/testdb","sa", "");`
  Niewielkie bazy danych mogą być uruchamiane do pracy w pamięci operacyjnej komputera:
  `Connection c=DriverManager.getConnection("jdbc:hsqldb:mem:testdb","sa","");`

  W obecnej wersji HSQLDB istnieje możliwość jednoczesnego używania wielu „serwerów” baz danych działających w trybie stand-alone.

- **Napisać program, który wysyła zapytanie do bazy danych i odbiera wyniki**

```
import java.sql.*;
public class Main {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "root";
        String password = "password";
        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM users");
            while (resultSet.next()) {
                System.out.println(resultSet.getInt("id") + " " + resultSet.getString("username"));}
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

Wykład 11 Refleksja

- **Jak uzyskać referencje do obiektu Class(Class c = (...))**
- **Jak uzyskać instancję klasy Class, co z typami prymitywnymi i tablicami typów prymitywnych**
  `c=boolean.class;` dla typów prymitywnych
  `c=”jakisstring”.getClass();` jeśli mamy instancję klasy
  `byte[] bytes, c=bytes.getClass();` wypisze ładnie [B
- **Proxy (do czego służy, jak stworzyć, czego potrzebuje)**
  W programowaniu refleksyjnym, "proxy" to wzorzec projektowy, który umożliwia tworzenie obiektów, które działają jako pośrednicy dla innych obiektów. Proxy jest obiektem, który działa jak inny obiekt, ale umożliwia kontrolowanie i dostosowywanie zachowania tego obiektu. W kontekście programowania refleksyjnego, proxy jest obiektem, który implementuje interfejs klasy lub interfejsu, który chcemy przeproksować. Gdy klasa lub interfejs jest przeproksowany, każde wywołanie metody na tym obiekcie zostanie przechwycone i przekierowane do metody w proxy. Ta metoda może dodać specjalne zachowanie przed lub po wywołaniu właściwej metody, lub zastąpić całkowicie oryginalną metodę.Proxy może być wykorzystywane do wielu zadań, takich jak:

  - Zapewnienie **bezpieczeństwa**: Proxy może zapewnić dodatkowe zabezpieczenia, takie jak uwierzytelnianie użytkowników, przed wywołaniem właściwej metody.
  - **Łączenie zdalne:** Proxy może służyć do łączenia się z obiektami, które znajdują się na innej maszynie i zapewnić komunikację między nimi.
  - **Logowanie**: Proxy może służyć do logowania wywołań metod, co umożliwia późniejszą analizę działania aplikacji.
  - **Kontrola dostępu:** Proxy może służyć do kontroli dostępu do metod, np. poprzez sprawdzenie uprawnień użytkownika przed wywołaniem metody.

  Ogólnie rzecz biorąc, proxy jest użytecznym narzędziem w programowaniu refleksyjnym, które umożliwia kontrolowanie i dostosowywanie zachowania obiektów na wiele różnych sposobów.

```
InvocationHandler handler = new MyInvocationHandler(...);
Class proxyClass = Proxy.getProxyClass(
 MyInterface.class.getClassLoader(),
  new Class[] { MyInterface.class });
MyInterface mi = (MyInterface) proxyClass.getConstructor(
 new Class[] { InvocationHandler.class }).newInstance(
 new Object[] { handler })
```

- **Invocation handler**

  W skrócie, InvocationHandler jest obiektem, który przechwytuje wywołania metod i definiuje, co ma się dziać z tymi wywołaniami. Jest to często używane wraz z mechanizmem proxy, który umożliwia tworzenie obiektów pośredniczących, które przechwytują wywołania metod do innych obiektów. Kiedy obiekt proxy otrzymuje wywołanie metody, przekazuje ono swojemu obiektowi InvocationHandler, który decyduje, co zrobić z tym wywołaniem. InvocationHandler może wywołać metodę na oryginalnym obiekcie, zmienić argumenty metody lub całkowicie zastąpić wywołanie.

- **Wypisać refleksyjnie metody klasy Object**

```
Class c = Class.forName(className);
  for(Method m: c.getMethods()){
      System.out.println(m.getName());
   }
```

- **Dal kod i trzeba bylo to napisac za pomoca refleksji (jedna klasa)**

```
public static void main(String[] args) {
        try {
            ReflectExample example = new ReflectExample();
            Class<?> cls = example.getClass();
            Method method = cls.getDeclaredMethod("reflectMethod", String.class);
            method.invoke(example, "Hello");//podajemy obiekt,na którym wyw. metode
        } catch (Exception e) {             //a nastepnie argumenty
            e.printStackTrace();
        }
    }
}
```

Wykład 12 Dynamiczne

- **Jak dostać instancje klasy z bytecodu (classloader itd)**
- Jak załadować klasę z tablicy bajtów (`ClassLoader::defineClass`)

```
public class MyClassLoader  {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccessException, NoSuchMethod
        ClassLoader cl = ClassLoader.getSystemClassLoader();
        Class loadedClass = cl.loadClass("Klasa");
        Constructor[] constructor = loadedClass.getConstructors();
        Klasa obj = (Klasa)constructor[0].newInstance();
        System.out.println(obj.pole);//spr
    }
}
```

- **ASM**
  ASM to biblioteka ułatwiająca manipulację bytecodem Javy,przykładowy kod generujący klasę Main wygląda następująco:
  `(java -cp asm-4.1.jar:asm-util-4.1.jar:asm-commons-4.1.jar org.objectweb.asm.util.ASMifier Main.class)`
- **Struktura pliku class i co się w tym pliku znajduje**
- **Co oznaczają 4 pierwsze bajty w kazdej z klas w javie(czy jakos tak)**
  W języku Java każda klasa rozpoczyna się nagłówkiem o stałej długości 8 bajtów, zwanej "magicznym numerem" **(magic number)**. "Magiczny numer" składa się z pierwszych czterech bajtów i jest to stała wartość, która identyfikuje plik jako plik klasy Java.
  `CA FE BA BE – “magic” - identyfkator formatu pliku`
  Kolejne cztery bajty nagłówka zawierają **wersję formatu pliku klasy**. Pierwszy z tych czterech bajtów określa numer głównej wersji, a drugi numer wersji podrzędnej. Dwa pozostałe bajty określają numer wydania, który zwykle jest zawsze ustawiony na wartość

zero.Kolejne cztery to `00 22` – ilość deklarowanych elementów (Constant Pool). Po tej deklaracji następują kolejne, 33 elementy.

- **Bajtkod, jak uruchomić metode**
  Do tego służy metoda `invokevirtual`

Wykład 13 Funkcyjne

- **Czym jest java.util.Function, co zawiera, jakie ma metody**
  Zawiera metody: Function, Bifunction, UnaryOperator, Suplier, Consumer, Predicate
  Interfejs funkcyjny to interfejs, ktory zawiera *jedną* metodę abstrakcyjną, którą należy zaimplementować.
  W programowaniu funkcyjnym funkcje powinny działać zawsze tak samo i nie powinny powodować żednych “efektów ubocznych”.
  Podstwowe zasady:
  ● wszystkie zmienne są stałe (final),
  ● nie ma zmiennych globalnych,
  ● funkcje mogą być zarówno argumentami innych funkcji oraz mogą
  być zwracane jako wynik innych funkcji.
- **Czym są Supplier i Consumer**
  `Consumer<Integer> cons = x->{System.out.println(x);};` Consumer dostaje jeden argument i nic nie zwraca
  `Supplier<Integer> sup = () -> 7;` Suplier nie dostaje argumentów i zwraca jeden wynik, jedna metoda get()
  `Runnable r = ()->{System.out.println("Hello");};`
- **Predykat - czym jest itp itd**, potem przeszedł na temat Function i kazał napisać przykładową funkcję
  Predicate to funkcja, która zwraca boolean, w test możemy umiścić np. warunek, którego potem użyjemy w filter

```
Predicate<T>
 boolean test(T t);
 Predicate<T> and(Predicate<? super T>, other);
 Predicate<T> or(Predicate<? super T>, other);
 Predicate<T> negate();
```

- **Co to unary operator?**
  UnaryOperator to funkcja, której wynik jest tego samego typu co argument

```
public interface UnaryOperator<T> extends Function<T,T>
example: UnaryOperator<Integer> func2 = x -> x * 2;
```

- **napisz w jednej linijce program, który wypisze za pomocą strumieni wszystkie elementy vecobj z poprzedniego pytania**

```
Stream.of(vecobj).forEach(Systemp.out::println);
```

- (x, y) -> x * y; co to jest (samo stwierdzenie że Function nie wystarczyło, albowiem chodziło o Bifunction
- **przefiltrować arrayliste integerów zeby zostaly parzyste**

```
List<Integer> list1 = Arrays.asList(1, 2, 3);
//List<Integer> list2 = Arrays.asList(4, 5, 6);
Stream.of(list1) // Stream<List<Integer>>
 //.flatMap(List::stream) // Stream<Integer>
 .filter(num -> num % 2 == 0) // zostają liczby parzyste
 .forEach(System.out::println); // wskaźnik do funkcji inaczej:
                                // num->System.out.println(num)
```

- **zrobić, żeby sie kompilowalo : b = x->x+1**
  `Function<Integer,Integer> b = x→x+1;` ??