Call-back идеология в Java

Създаването на по-гъвкави класове понякога предполага възможности за подаване на функция-параметър към някой от методите. В предходниците на Java (C и C++) това може да се постигне чрез предаване на параметър-указател към потребителска функция (callback). Поради принципа за защитеност, в Java-кода такава възможност е избегната. Това не значи, че тази гъвкавост не може да се постигне.

Реализация чрез наследяване

Най-простата идея за реализация на такава гъвкавост е чрез наследяване: авторът на класа предвижда публичен метод, който ще се вика динамично при възникване на желаното събитие, а ползвателят прави наследник на класа, в който предефинира (само) този метод според своите нужди. (Този метод може и да е абстрактен, което ще принуди ползвателя да го дефинира.)

ПРИМЕРЕН ПРОЕКТ: „Краен автомат“

Модул за класа Automaton

```
import java.util.ArrayList;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Automaton {
 private int posCnt;//Брой състояния
 private ArrayList<Integer> finPos=new ArrayList<Integer>();//Списък от крайни състояния
 private ArrayList<Arrow> arr=new ArrayList<Arrow>(); //Списък от стрелки
 class Arrow{//Клас "Стрелка"
  private int from,to; //От-до
  private char with;   //буква от азбуката върху стрелката
  //Конструктор
  public Arrow(int f,int t,char w){
   from=f;
   to=t;
   with=w;
  }
  //Гетъри
  public int getFrom(){
   return from;
  }
  public int getTo(){
   return to;
  }
  public char getWith(){
   return with;
  }
 @Override
  public String toString(){
   return from+"--"+with+"->"+to;//1--C->5
  }
 }
 //Конструктор с данни от файл
 public Automaton(String fName) throws FileNotFoundException {
  try{
   File file = new File(fName);
   Scanner inp = new Scanner(file);
   int K,M,p;
   posCnt=inp.nextInt();
   K=inp.nextInt();
   for (int i=0;i<K;i++){
    p=inp.nextInt();
    finPos.add(p);
   }
   M=inp.nextInt();
   int f,t; char w;
   for (int i=0;i<M;i++){
      f=inp.nextInt();
      t=inp.nextInt();
      w=inp.next().charAt(0);
      arr.add(new Arrow(f,t,w));
   }
   inp.close();
```

```
  }catch(FileNotFoundException e){
      throw new FileNotFoundException();
  }
 }
 //Гетъри
 public int getPosCnt(){
  return posCnt;
 }
 public int getFinPosCnt(){
  return finPos.size();
 }
 public int getArrCnt(){
  return arr.size();
 }
 public Arrow getArrow(int n){
  return arr.get(n);
 }
 @Override
 public String toString(){
  String s="N="+posCnt+", K="+getFinPosCnt()+":{ ";
  for (int i:finPos) s=s + i + " ";
  s=s+"}, M="+getArrCnt()+": { ";
  for (Arrow a:arr) s+=a+" ";
  s=s + "}";
  return s;
 }
 public boolean isFinal(int p){
  return finPos.contains(p);
 }
 private boolean trace(int p,String w){
  if (w.isEmpty()) return isFinal(p);
  for (int i=0;i<getArrCnt();i++)
   if (getArrow(i).getFrom()==p && getArrow(i).getWith()==w.charAt(0))
      if (trace(getArrow(i).getTo(),w.substring(1))) return true;
  return false;
 }
 public boolean isWord(String w){
  return trace(0,w);
 }
 public void proceed(String s){ //Метод, който се вика при възникване на желана ситуация
  System.out.println(s); //Действие по подразбиране
                         //(в случая - извеждане на стандартния изход
 }
 private void make(int len,int p,String w){
  if (w.length()>len) return;
  if (w.length()==len){
   if (isFinal(p)) proceed(w); //Повикване на метода
   return;
  }
  for (int i=0;i<getArrCnt();i++)
   if (getArrow(i).getFrom()==p) make(len,getArrow(i).getTo(),w+getArrow(i).getWith());
 }
 public void makeWords(int len){
  make(len,0,"");
 }
}
```

Потребителски модул за тестване - клас Main

```
import java.io.FileNotFoundException;
class Automaton1 extends Automaton{ //Деклариране на наследник
 private int n; //Ново свойство (брояч на думите)
 //Конструктор - като наследения
 public Automaton1(String fName) throws FileNotFoundException{
  super (fName);
  n=0; //Добавяме инициализация на свойството
 }
```

```
 @Override //При желание – предефинираме обработващата функция
 public void proceed(String s){
  super.proceed(s); //(Направи като наследения метод…
  n++;              // и увеличи брояча с едно.)
 }
 public int getN(){  //Гетър за новото свойство
  return n;
 }
}
public class Main {
 public static void main(String[] args) {
  try{
   Automaton1 a=new Automaton1("aut1.txt");
   a.makeWords(7);
   System.out.println(a.getN());
  }catch(Exception e){
    System.out.println("File not found");
  }
 }
}
```

Реализация чрез интерфейс

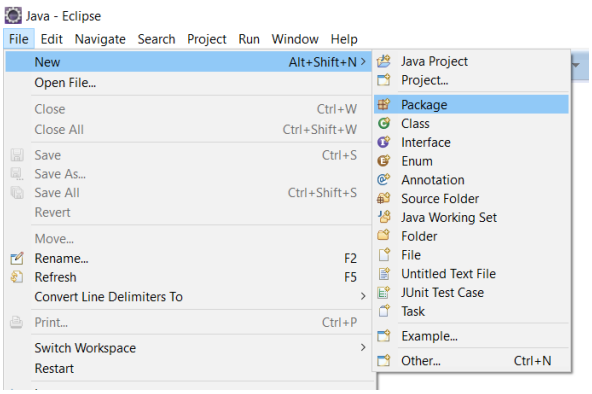
Java поддържа и идеологията „интерфейс“, с помощта на която също може да се постигне желаната гъвкавост. Просто потребителят на класа може да създаде клас, който използва (*implements*) интерфейс, в който реализира метода, който той иска да подаде във вид на параметър. От своя страна разработчикът на класа оформя нужния интерфейс и декларира, че той се използва в основния разработван клас. Тази идея не е нова: да си спомним интерфейса *Comparator*, който много удобно употребяваме в рамката **Collections** (за сортиране, например).

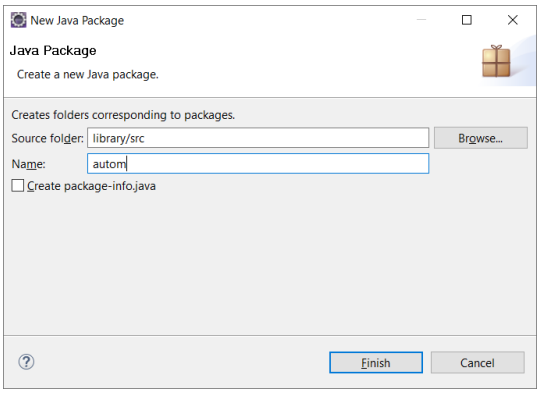
За целта е най-стилно да се създаде ***потребителска библиотека***, която съдържа основния разработван клас и (поне) интерфейса, който дефинира (и, евентуално, реализира) функцията-параметър.

ПРИМЕРЕН ПРОЕКТ: „Краен автомат“

Първо ще създадем библиотеката **autom**, която съдържа класа Automaton и интерфейса Callback. За целта в Eclipse:

1. Отваряме нов Java-проект и го наричаме, например, library (тук предпочитайте за името **малки** латински букви). Тази папка по-нататък можем да ползваме и за други потребителски библиотеки.
2. Към проекта отваряме „библиотека“: **Package** (отново за името се предпочитат малки латински букви, тук ще наречем библиотеката autom).





3. Добавяме двата библиотечни класа Callback и Automaton:

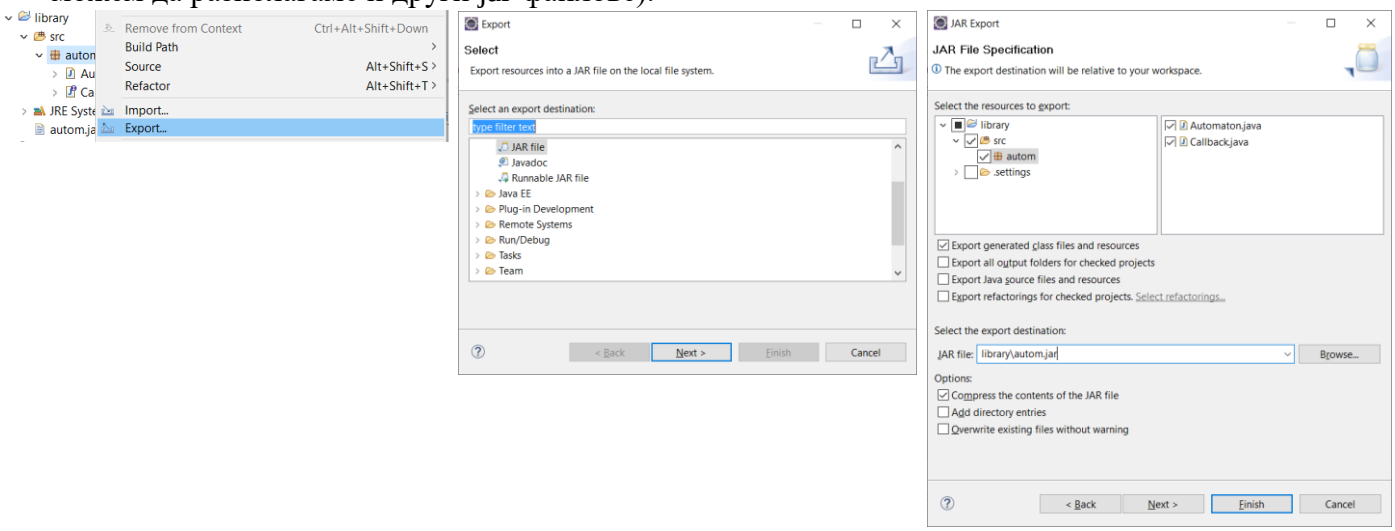
```
package autom; //Име на библиотеката
//Функция-параметър (callback function), която ще се повиква при възникване на важното
// събитие (в случая – създаване на нова дума от езика)
public interface Callback {
 default public void proceed(String s){
  System.out.println(s);//По подразбиране – изход на стандартния изход,
                        // следван от нов ред
 }
}
```

```
package autom; //Име на библиотеката
//Ползвани библиотеки
import java.util.ArrayList;
import java.io.File;
import java.io.FileNotFoundException;
```

```java
import java.util.Scanner;
public class Automaton implements Callback {  //Тук обявяваме
                                               // връзката с интерфейса
 private int posCnt;//Брой състояния
 private ArrayList<Integer> finPos=new ArrayList<Integer>();//Списък от крайни
                                                            // състояния
 private ArrayList<Arrow> arr=new ArrayList<Arrow>(); //Списък от стрелки
 class Arrow{//Клас "Стрелка". Този клас спокойно може да е вътрешен
  private int from,to; //От-до
  private char with;   //буква от азбуката върху стрелката
  //Конструктор
  public Arrow(int f,int t,char w){
   from=f;
   to=t;
   with=w;
  }
  //Гетъри
  public int getFrom(){
   return from;
  }
  public int getTo(){
   return to;
  }
  public char getWith(){
   return with;
  }
 @Override
  public String toString(){
   return from+"--"+with+"->"+to;//1--C->5
  }
 }
 //Конструктор с данни от файл
 public Automaton(String fName) throws FileNotFoundException {
  try{
   File file = new File(fName);
   Scanner inp = new Scanner(file);
   int K,M,p;
   posCnt=inp.nextInt();
   K=inp.nextInt();
   for (int i=0;i<K;i++){
    p=inp.nextInt();
    finPos.add(p);
   }
   M=inp.nextInt();
   int f,t; char w;
   for (int i=0;i<M;i++){
      f=inp.nextInt();
      t=inp.nextInt();
      w=inp.next().charAt(0);
      arr.add(new Arrow(f,t,w));
   }
   inp.close();
  }catch(FileNotFoundException e){
      throw new FileNotFoundException();
  }
 }
 //Гетъри
 public int getPosCnt(){
  return posCnt;
 }
 public int getFinPosCnt(){
  return finPos.size();
 }
 public int getArrCnt(){
  return arr.size();
 }
 public Arrow getArrow(int n){
  return arr.get(n);
 }
```

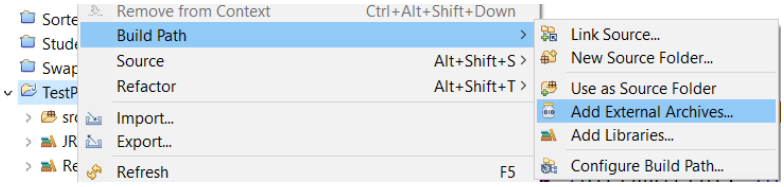
```
 @Override
 public String toString(){
  String s="N="+posCnt+", K="+getFinPosCnt()+":{ ";
  for (int i:finPos) s=s + i + " ";
  s=s+"}, M="+getArrCnt()+": { ";
  for (Arrow a:arr) s+=a+" ";
  s=s + "}";
  return s;
 }
 public boolean isFinal(int p){
  return finPos.contains(p);
 }
//Частна рекурсивна трасираща функция
 private boolean trace(int p,String w){
  if (w.isEmpty()) return isFinal(p);
  for (int i=0;i<getArrCnt();i++)
   if (getArrow(i).getFrom()==p && getArrow(i).getWith()==w.charAt(0))
      if (trace(getArrow(i).getTo(),w.substring(1))) return true;
  return false;
 }
//Метод за решаване на правата задача с очакван от потребителя интерфейс
 public boolean isWord(String w){
  return trace(0,w);
 }
//Частна рекурсивна изграждаща функция
//Вход: желана дължина len, текуща позиция в автомата p, натрупана досега дума w,
//      обект от тип Callback, който реализира метода за обработка proceed
 private void make(int len,int p,String w,Callback c){
  if (w.length()>len) return;//Думата е станала дълга - "изход с неуспех"
  if (w.length()==len){//Думата е дълга точно колкото трябва:
   if (isFinal(p)) c.proceed(w);//обработка, ако p е крайно състояние
   return;// и отново изход
  }
  for (int i=0;i<getArrCnt();i++)
   if (getArrow(i).getFrom()==p) //напред по следващата стрелка, която започва от
                                 // p, натрупвайки символа, който е "върху" нея.
    make(len,getArrow(i).getTo(),w+getArrow(i).getWith(),c);
 }
//Метод с удобен за потребителя интерфейс: каква дължина иска и с каква
// (негова, на потребителя на класа!) функция да се обработва
 public void makeWords(int len,Callback c){
  make(len,0,"",c);
 }
}
```

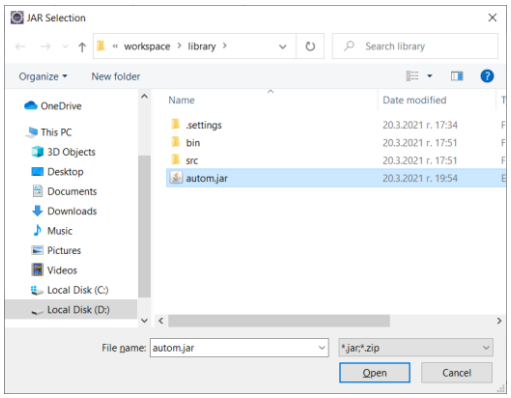
4. Следващата стъпка представлява експорт на написаната библиотека в Java-архивен (jar) файл и записването му на определеното от нас място (в случая ще избера директно папката library, където можем да разполагаме и други jar-файлове).



5. Сега библиотеката е готова за използване, проектът library може да бъде затворен. Ето един примерен проект TestPrj, който дава представа как се използва създадената библиотека.

- Създаваме Java-проекта с име TestPrj, който включва клас Main с тестов метод main.
- Добавяме пътя за достъп до библиотеката autom.jar в папката library:





- Евентуално създаваме в папката TestPrj необходим ресурс (в случая – текстовия файл aut1.txt, който описва автомат според възприетия формат).
- Пишем потребителския код:

```
import autom.*;//Включване на библиотеката с всички създадени в нея класове
//Клас за предефиниране на обработващата функция proceed
// (в случая – искаме изходът да бъде предхождан от брояч)
class Callback1 implements Callback{
 static int n=0;
 @Override
 public void proceed(String s) {
  System.out.print(++n+": ");
  Callback.super.proceed(s);
 }
}
public class Main {
 public static void main(String[] args) {
  try{
   Automaton a=new Automaton("aut1.txt");
   //Направи всички седембуквени думи, описани чрез автомата, като всяка дума бъде
   // обработена по желания от мен начин (а именно – извеждана на стандартния изход и
   // предхождана от номер)
   a.makeWords(7, new Callback1());
  }catch (Exception e){
   System.out.println("File not found");
  }
 }
}
```

Така ползвателят на библиотеката отново контролира ВСИЧКО: кой автомат да се създаде и КАК ДА СЕ ОБРАБОТВА всяка получена нова дума.