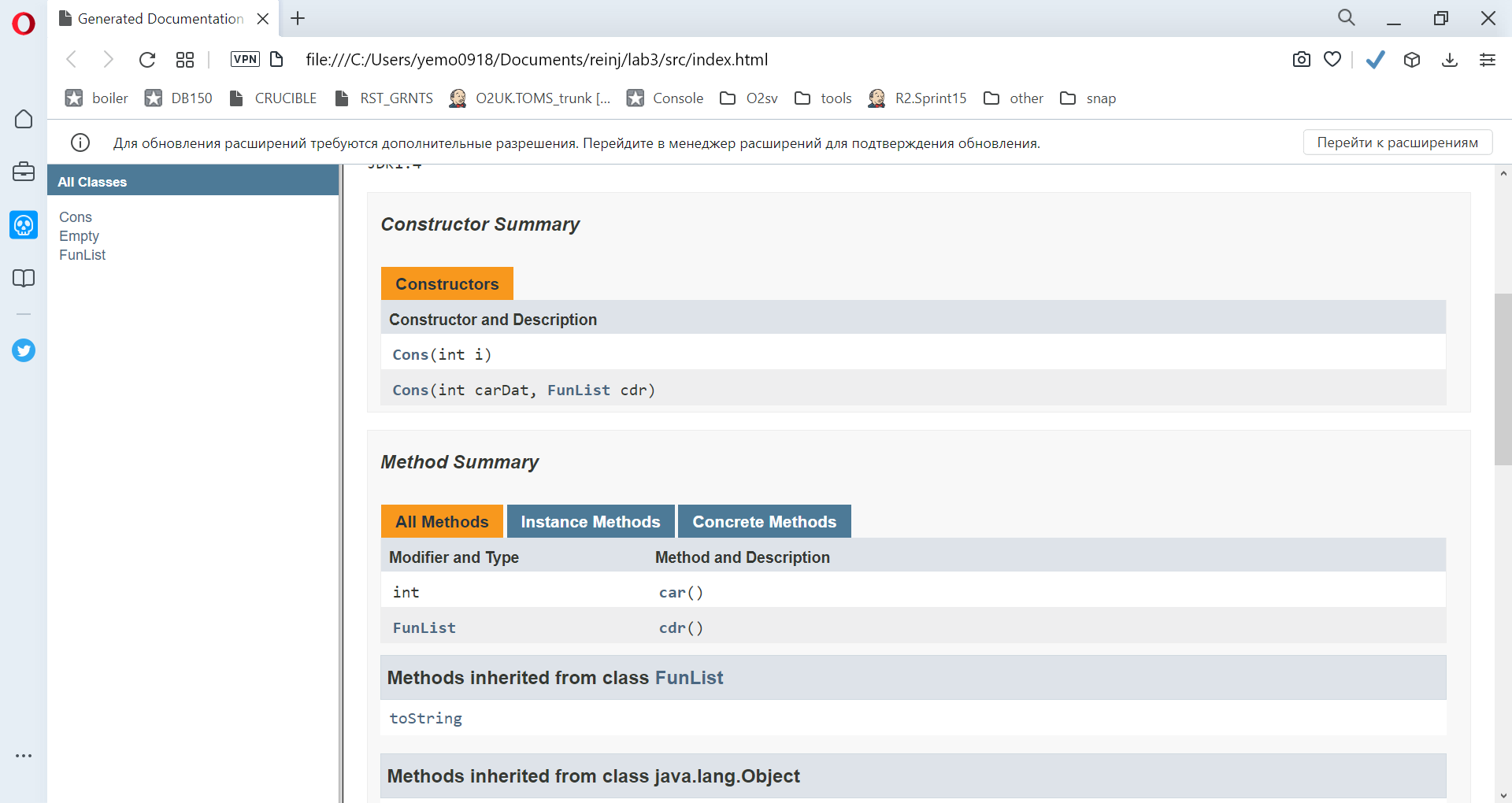
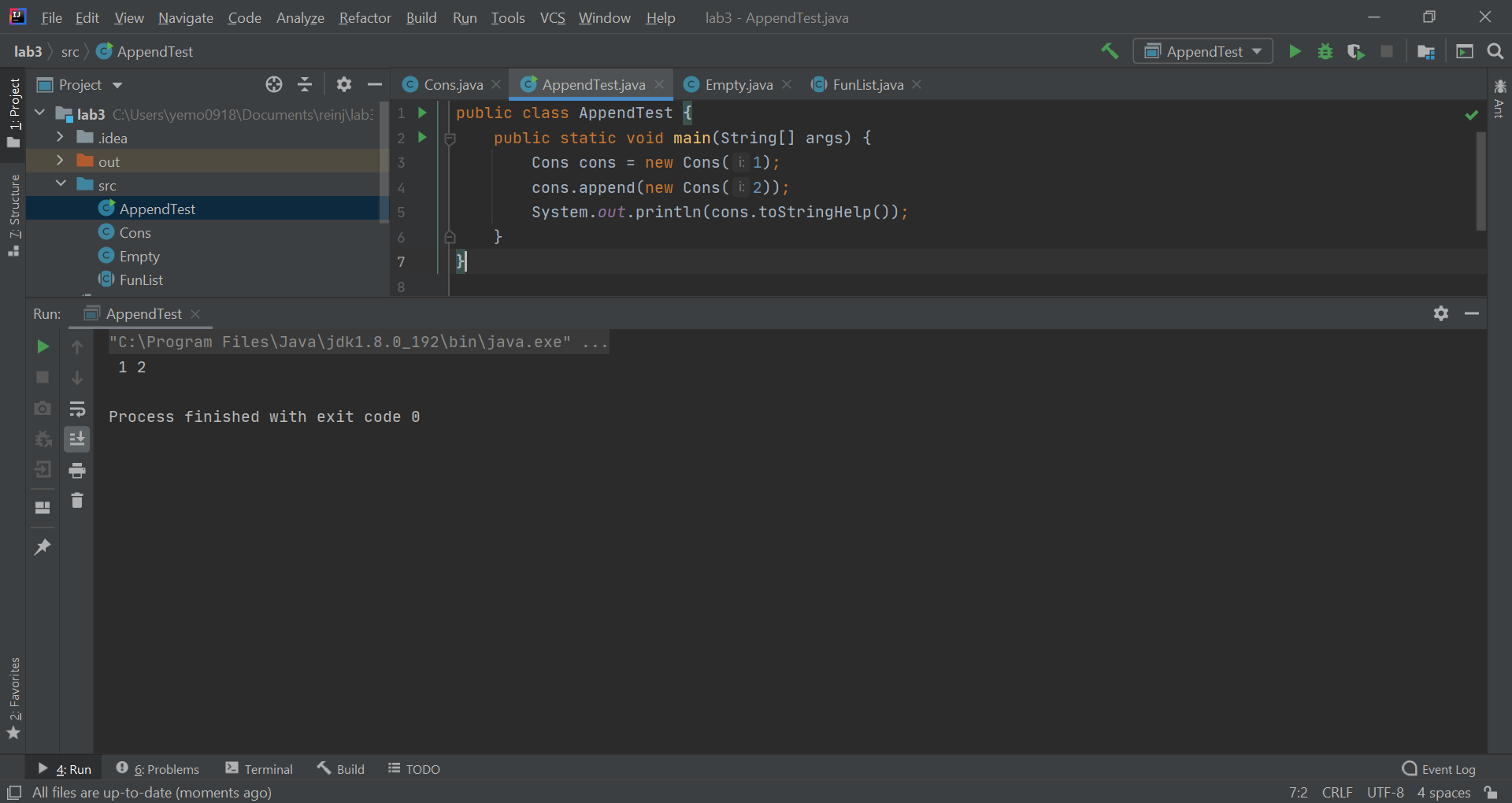
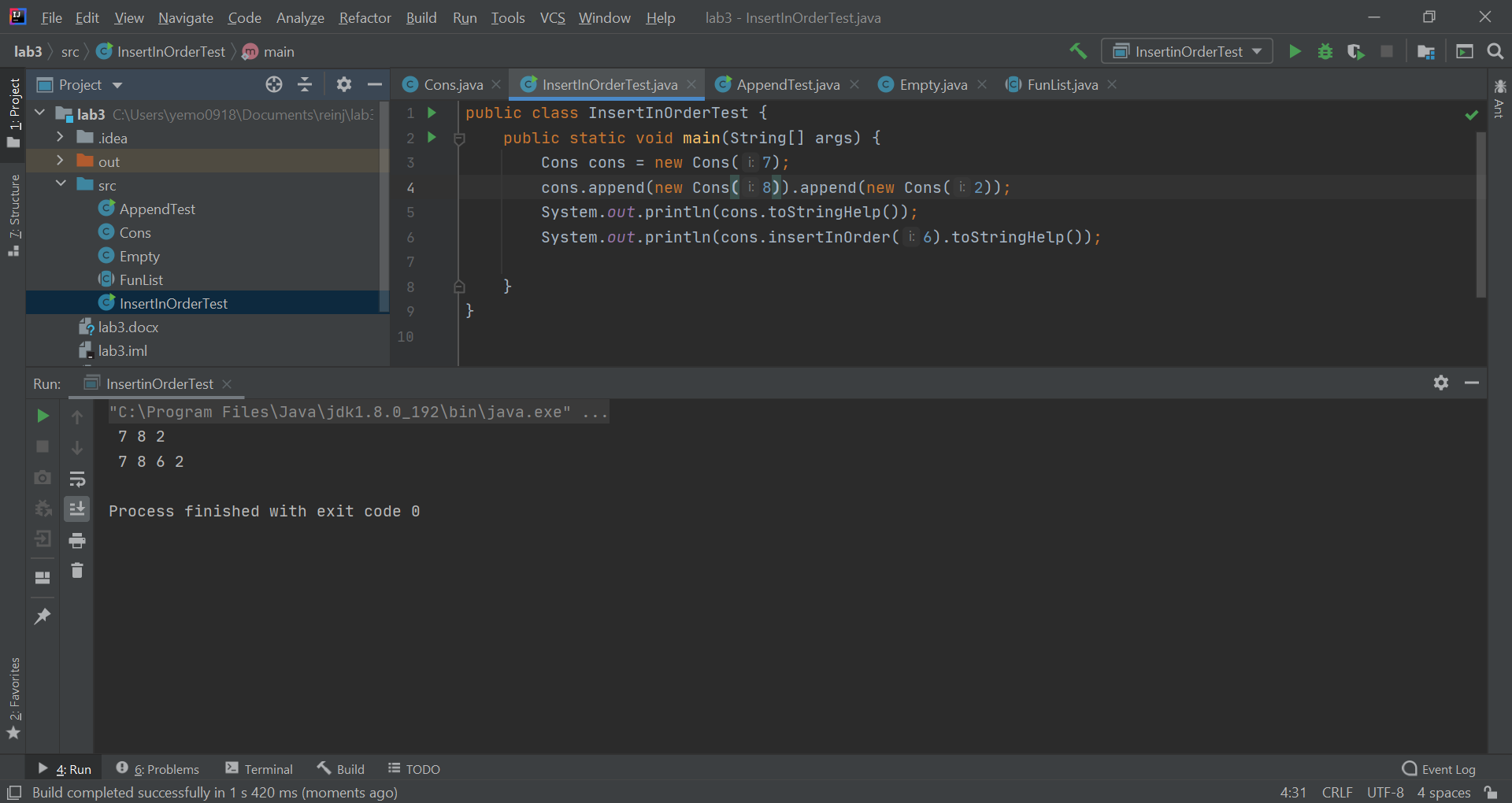
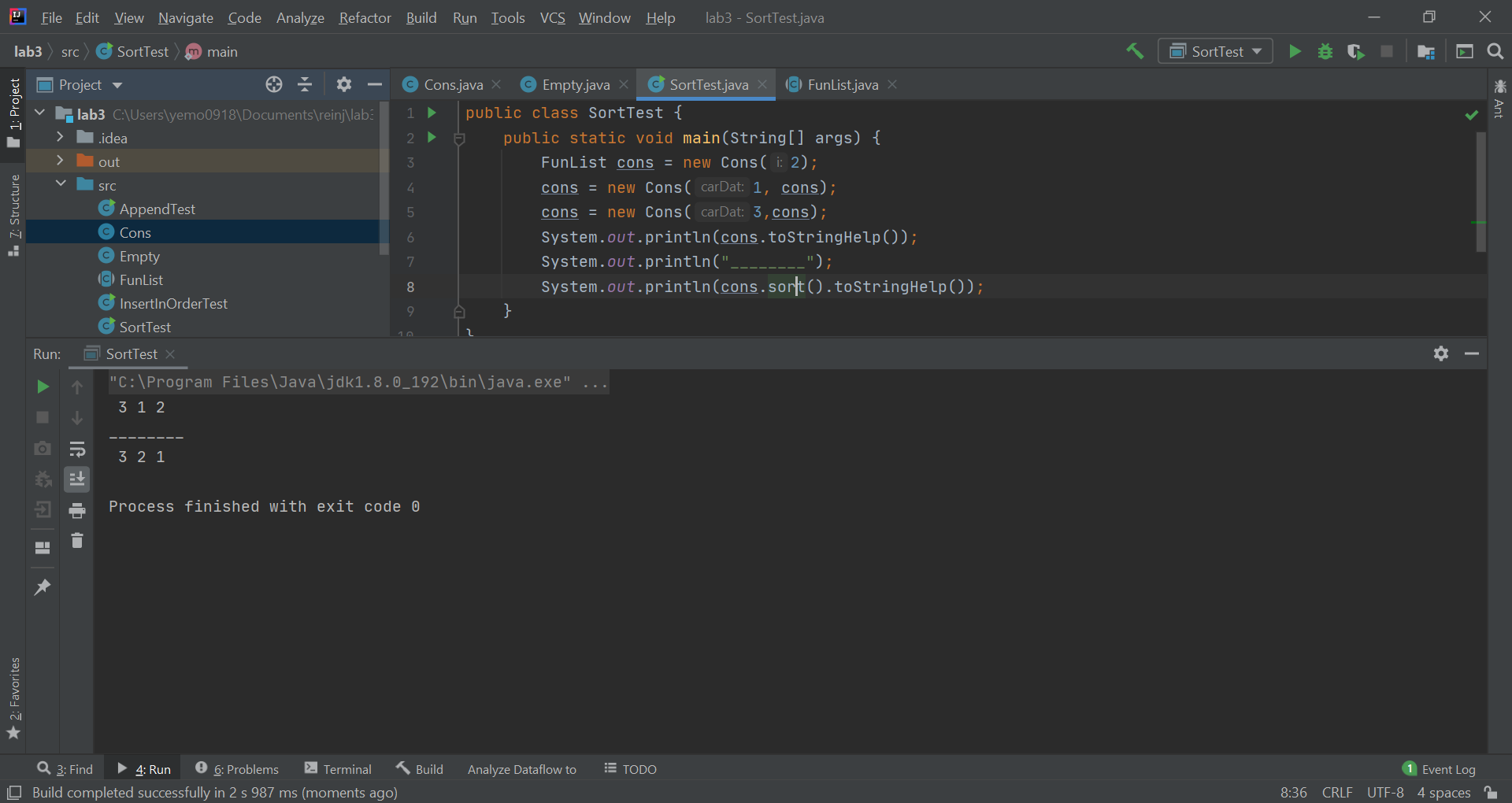
2.1  


2.2







import java.util.Collections;  
import java.util.LinkedList;  
  
*/\*\*  
 \* Mimics fundamental function cons for constructs  
 \* memory objects which hold two values or pointers to values  
 \*  
 \** ***@author*** *Vyacheslav Moskalenko  
 \** ***@since*** *JDK1.4  
 \*/*public class Cons extends FunList {  
 private int \_dat;  
 private FunList \_cdr;  
  
 public Cons(int carDat, FunList cdr) {  
 \_dat = carDat;  
 \_cdr = cdr;  
 }  
  
 */\*\*  
 \** ***@param*** *i a left side (head) of the list.  
 \*/* public Cons(int i) {  
 \_dat = i;  
 \_cdr = Empty.*uniqueInstanse*();  
 }  
  
 */\*\*  
 \** ***@return*** *the first int in the list object  
 \*/* public int car() {  
 return \_dat;  
 }  
  
 */\*\*  
 \** ***@return*** *the tail (all but the first element) of the list object  
 \*/* public FunList cdr() {  
 return \_cdr;  
 }  
  
 @Override  
 public FunList append(FunList other) {  
 if(this.\_cdr instanceof Empty){  
 return new Cons(this.\_dat, other);  
 }  
 return new Cons(this.\_dat, this.\_cdr.append(other));  
 }  
  
 @Override  
 public FunList insertInOrder(int i) {  
 if(\_dat<=i) {  
 return new Cons(i, this);  
 }else{  
 return new Cons(\_dat, \_cdr.insertInOrder(i));  
 }  
  
 }  
  
 @Override  
 public FunList sort() {  
 return \_cdr.sort()  
 .insertInOrder(\_dat);  
 }  
  
 */\*\*  
 \** ***@return*** *a String description of the list object  
 \*/* String toStringHelp() {  
 return " " + \_dat + \_cdr.toStringHelp();  
 }  
}

*/\*\*  
\* Mimics functional empty list.  
\** ***@author*** *Vyacheslav Moskalenko  
\** ***@since*** *JDK1.4  
\*/*public class Empty extends FunList{  
  
private static Empty *EMPTY* = new Empty();  
  
*/\*\*  
\** ***@return*** *single instance of class  
 \*/*public static Empty uniqueInstanse(){  
 return *EMPTY*;  
}  
  
private Empty(){}  
  
*/\*\*  
\** ***@return*** *nothing now  
 \*/*public int car(){  
throw new java.util.NoSuchElementException("car requires a non Empty Funlist");  
}  
  
*/\*\*  
\** ***@return*** *nothing now  
 \*/*public FunList cdr(){  
throw new java.util.NoSuchElementException("cdr requires a non Empty Funlist");  
}  
  
 public FunList append(FunList other) {  
 return new Cons(this.car(), other);  
 }  
  
  
 public FunList insertInOrder(int i) {  
 return new Cons(i);  
 }  
  
  
 public FunList sort() {  
 return this;  
 }  
  
  
  
 */\*\*  
\** ***@return*** *empty string  
 \*/*String toStringHelp(){  
return "";  
}  
  
  
}

*/\*\*  
 \* Mimics functional lists.  
 \*  
 \** ***@author*** *Moskalenko Vyacheslav  
 \** ***@since*** *JDK1.4  
 \*/*public abstract class FunList {  
 */\*\*  
 \** ***@return*** *the first int in the list object  
 \*/* public abstract int car();  
  
 */\*\*  
 \** ***@return*** *the tail (all but the first element) of the list object  
 \*/* public abstract FunList cdr();  
  
 */\*\*  
 \* NOTE: toString () method is NOT abstract. It calls, toStringHelp() , an abstract method.  
 \* It represents what we call an "invariant" behavior for <code>FunList</code>.  
 \* It is an example of the "Template Method Pattern". A "template method" is a method that  
 \* makes calls to at least one abstract method in its own class.  
 \*/* public String toString() {  
 return "(" + toStringHelp() + " ) ";  
 }  
  
 */\*\*  
 \** ***@return*** *some FunList  
 \*/* public abstract FunList append(FunList other);  
  
 public abstract FunList insertInOrder(int i);  
  
 public abstract FunList sort();  
  
  
 */\*\*  
 \** ***@return*** *a String description of the list object  
 \*/* abstract String toStringHelp();  
}