**Perangkat Lunak untuk Menguji Grammar dalam Bahasa Java**

**Mata Kuliah IF5020 - Algoritma dan Pemrograman A**

Oleh :

1. Sukamto - 23518017
2. Andreas Novian Dwi T. - 23518002

**Contoh masukan dan keluaran**

Tiga contoh input file teks beserta keluarannya:

1. Input1.txt

|  |
| --- |
| package Model;  import Controller.Controller;  import java.io.IOException;  public abstract class Animal {  public String hello;  public int age, weight;  public final static int MOUTH = 1;    public Animal(int age, int weight){  this.age = age;  this.weight = weight;  age += 5;  int a= "b".length();  String hallo = new String();  hello = new String();  } |

Keluaran :

|  |
| --- |
| package Model;  import Controller.Controller;  import java.io.IOException;  public abstract class Animal{  public Stringhello;  public int age,weight;  public final static int MOUTH=1;  public Animal(int age,int weight){  this.age=age;  this.weight=weight;  age+= 5;  int a="b".length();  Stringhallo=new String();  hello=new String();  }  Tidak ada error |

1. Input2.txt

|  |
| --- |
| public class Tester {  public static void main(String[] args) throws IOException {  Controller ct = new Controller("input1.txt","output1.txt");  ct.start();  ct = new Controller("input2.txt","output2.txt");  ct.start();  ct = new Controller("input3.txt","output3.txt");  ct.start();  }  } |

Keluaran :

|  |
| --- |
| public class Tester{  public static void main(String[]args)throws IOException{  Controllerct=new Controller("input1.txt","output1.txt");  ct.start();  ct=new Controller("input2.txt","output2.txt");  ct.start();  ct=new Controller("input3.txt","output3.txt");  ct.start();  }  }  Tidak ada error |

1. Input3.txt

|  |
| --- |
| public class Number {  public int hello(int age, int weight){  int z = {1,2,3,4};  age += 5;  }  }  package Model;  import Controller.Controller;  import java.io.IOException; |

Keluaran :

|  |
| --- |
| public class Number{  public int hello(int age,int weight){  int z={  1,2,3,4}  ;  age+= 5;  }  }  (Error)packageModel;importController.Controller;importjava.io.IOException; |

**Kode Program**

**Kelas ProgramDeclaration.java**

|  |
| --- |
| package Model;  import Controller.Controller;  import java.io.IOException;  /\*\*  \* @author Sukamto 23518017 Andreas Novian 23518002  \*/  public class ProgramDeclaration {    Controller cnt;    public ProgramDeclaration(Controller cont) {  this.cnt = cont;  }  //Program Declaration  public void compilationUnit() throws IOException {  if(cnt.symbol.equals("package")){  packageDeclaration();  while (cnt.symbol.equals("import")) {  importDeclaration();  }  while (cnt.symbol.equals("public") || cnt.symbol.equals("final") || cnt.symbol.equals("class") || cnt.symbol.equals("interface") || cnt.symbol.equals("abstract")) {  classDeclaration();  }  } else if(cnt.symbol.equals("import") || cnt.symbol.equals("final")  || cnt.symbol.equals("class") || cnt.symbol.equals("interface")  || cnt.symbol.equals("abstract") || cnt.symbol.equals("public")){  while (cnt.symbol.equals("import")) {  importDeclaration();  }  while (cnt.symbol.equals("public") || cnt.symbol.equals("final") || cnt.symbol.equals("class") || cnt.symbol.equals("interface") || cnt.symbol.equals("abstract")) {  classDeclaration();  }  }  }  public void packageDeclaration() throws IOException {  this.cnt.accept("package");  packageName();  this.cnt.accept(";");  }  public void importDeclaration() throws IOException {  this.cnt.accept("import");  packageName();  importEnding();  }  public void packageName() throws IOException {  identifier();  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  identifier();  }  }  public void importEnding() throws IOException {  switch (cnt.symbol) {  case ("."):  this.cnt.accept(".");  this.cnt.accept("\*");  this.cnt.accept(";");  break;  case (";"):  this.cnt.accept(";");  break;  }  }  //Class  public void classDeclaration() throws IOException {  classModifier();  classModifierExtension();  }  public void classModifierExtension() throws IOException {  switch (cnt.symbol) {  case ("final"):  classModifier2();  this.cnt.accept("class");  classDeclarationExtension();  break;  case ("class"):  this.cnt.accept("class");  classDeclarationExtension();  break;  case ("abstract"):  abstractModifier();  typeDeclaration();  break;  case ("interface"):  this.cnt.accept("interface");  interfaceDeclaration();  break;  }  }  public void typeDeclaration() throws IOException {  switch (cnt.symbol) {  case ("class"):  this.cnt.accept("class");  abstractClassDeclaration();  break;  case ("interface"):  this.cnt.accept("interface");  interfaceDeclaration();  break;  }  }  public void classDeclarationExtension() throws IOException {  identifier();  super1();  interfaces();  classBody();  }  public void abstractClassDeclaration() throws IOException {  identifier();  super1();  interfaces();  abstractClassBody();  }  public void classModifier() throws IOException {  if (cnt.symbol.equals("public")) {  this.cnt.accept("public");  }  }  public void classModifier2() throws IOException {  if (cnt.symbol.equals("final")) {  this.cnt.accept("final");  }  }  public void abstractModifier() throws IOException {  this.cnt.accept("abstract");  }  public void super1() throws IOException {  if (cnt.symbol.equals("extends")) {  this.cnt.accept("extends");  identifier();  }  }  public void interfaces() throws IOException {  if (cnt.symbol.equals("implements")) {  this.cnt.accept("implements");  identifier();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  identifier();  }  }  }  public void classBody() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("public") || cnt.symbol.equals("protected") || cnt.symbol.equals("private") || cnt.symbol.equals("static")  || cnt.symbol.equals("transient") || cnt.symbol.equals("final") || cnt.symbol.equals("synchronized") || cnt.symbol.equals("volatile")  || cnt.symbol.equals("native") || cnt.symbol.equals("void") || cnt.symbol.equals("boolean")  || cnt.symbol.equals("float") || cnt.symbol.equals("double") || cnt.symbol.equals("byte") || cnt.symbol.equals("short")  || cnt.symbol.equals("int") || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  classBodyDeclaration();  }  this.cnt.accept("}");  }  public void classBodyDeclaration() throws IOException {  if (cnt.symbol.equals("public") || cnt.symbol.equals("protected") || cnt.symbol.equals("static")  || cnt.symbol.equals("transient") || cnt.symbol.equals("final") || cnt.symbol.equals("synchronized") || cnt.symbol.equals("volatile")  || cnt.symbol.equals("native") || cnt.symbol.equals("void") || cnt.symbol.equals("boolean")  || cnt.symbol.equals("float") || cnt.symbol.equals("double") || cnt.symbol.equals("byte") || cnt.symbol.equals("short")  || cnt.symbol.equals("int") || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  abstractMethodModifier();  bodyDeclaration();  } else if (cnt.symbol.equals("private")) {  this.cnt.accept("private");  abstractBodyDeclaration();  }  }  public void abstractMethodModifier() throws IOException {  switch (cnt.symbol) {  case ("public"):  this.cnt.accept("public");  break;  case ("protected"):  this.cnt.accept("protected");  break;  }  }  public void abstractClassBody() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("public") || cnt.symbol.equals("protected") || cnt.symbol.equals("private") || cnt.symbol.equals("static")  || cnt.symbol.equals("transient") || cnt.symbol.equals("final") || cnt.symbol.equals("synchronized") || cnt.symbol.equals("volatile")  || cnt.symbol.equals("native") || cnt.symbol.equals("abstract") || cnt.symbol.equals("void") || cnt.symbol.equals("boolean")  || cnt.symbol.equals("float") || cnt.symbol.equals("double") || cnt.symbol.equals("byte") || cnt.symbol.equals("short")  || cnt.symbol.equals("int") || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  abstractClassBodyDeclaration();  }  this.cnt.accept("}");  }  public void abstractClassBodyDeclaration() throws IOException {  if (cnt.symbol.equals("public") || cnt.symbol.equals("protected") || cnt.symbol.equals("static")  || cnt.symbol.equals("transient") || cnt.symbol.equals("final") || cnt.symbol.equals("synchronized") || cnt.symbol.equals("volatile")  || cnt.symbol.equals("native") || cnt.symbol.equals("abstract") || cnt.symbol.equals("void") || cnt.symbol.equals("boolean")  || cnt.symbol.equals("float") || cnt.symbol.equals("double") || cnt.symbol.equals("byte") || cnt.symbol.equals("short")  || cnt.symbol.equals("int") || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  abstractMethodModifier();  bodyDeclaration();  } else if (cnt.symbol.equals("private")) {  this.cnt.accept("private");  abstractBodyDeclaration();  }  }  public void bodyDeclaration() throws IOException {  if (cnt.symbol.equals("transient") || cnt.symbol.equals("volatile")) {  fieldDeclaration();  } else if (cnt.symbol.equals("native") || cnt.symbol.equals("synchronized") || cnt.symbol.equals("void")) {  methodInitializer();  } else if (cnt.symbol.equals("final")) {  this.cnt.accept("final");  finalDeclaration();  } else if (cnt.symbol.equals("static")) {  this.cnt.accept("static");  staticOption();  } else if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  dataTypeDeclaration();  }  }  public void dataTypeDeclaration() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  dataPrimitive();  dataPrimitiveDeclaration();  } else if (cnt.symbol.equals("\_") || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  declarationType();  }  }  public void dataPrimitiveDeclaration() throws IOException {  if (cnt.symbol.equals("\_") || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  variableOrMethodOption();  } else if (cnt.symbol.equals("[")) {  arrayAfterDataType();  variableLooping();  this.cnt.accept(";");  }  }  public void declarationType() throws IOException {  switch (cnt.symbol) {  case ("("):  constructorDeclaration();  break;  case ("throws"):  throws1();  break;  case ("="):  case (";"):  variableOperator();  variableLooping();  this.cnt.accept(";");  break;  case ("["):  arrayAfterDataType();  }  }  public void abstractBodyDeclaration() throws IOException {  if (cnt.symbol.equals("static") || cnt.symbol.equals("transient") || cnt.symbol.equals("final")  || cnt.symbol.equals("synchronized") || cnt.symbol.equals("volatile")  || cnt.symbol.equals("native") || cnt.symbol.equals("void") || cnt.symbol.equals("boolean")  || cnt.symbol.equals("float") || cnt.symbol.equals("double") || cnt.symbol.equals("byte") || cnt.symbol.equals("short")  || cnt.symbol.equals("int") || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  bodyDeclaration();  } else if (cnt.symbol.equals("abstract")) {  abstractMethodDeclaration();  }  }  public void finalDeclaration() throws IOException {  switch (cnt.symbol) {  case ("transient"):  fieldModifier3Declaration();  break;  case ("synchronized"):  finalAdditionalMod();  break;  }  }  public void staticOption() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  dataTypeDeclaration();  } else if (cnt.symbol.equals("final") || cnt.symbol.equals("void") || cnt.symbol.equals("synchronized")  || cnt.symbol.equals("transient") || cnt.symbol.equals("volatile") || cnt.symbol.equals("{")) {  staticDeclaration();  }  }  public void staticDeclaration() throws IOException {  if (cnt.symbol.equals("final") || cnt.symbol.equals("void")) {  staticMethodDeclaration();  block();  } else if (cnt.symbol.equals("{") || cnt.symbol.equals("transient") || cnt.symbol.equals("volatile")) {  staticInitializer();  } else if (cnt.symbol.equals("synchronized")) {  synchronizedModifier();  synchronizedMethodDeclaration();  }  }  public void staticInitializer() throws IOException {  if (cnt.symbol.equals("{")) {  block();  } else if (cnt.symbol.equals("transient") || cnt.symbol.equals("volatile")) {  staticFieldDeclaration();  }  }  //Interface  public void interfaceDeclaration() throws IOException {  identifier();  extendsInterfaces();  interfaceBody();  }  public void extendsInterfaces() throws IOException {  if (cnt.symbol.equals("extends")) {  this.cnt.accept("extends");  identifier();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  identifier();  }  }  }  public void interfaceBody() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("abstract") || cnt.symbol.equals("public")  || cnt.symbol.equals("static") || cnt.symbol.equals("final")) {  interfaceMember();  }  this.cnt.accept("}");  }  public void interfaceMember() throws IOException {  if (cnt.symbol.equals("abstract")) {  abstractMethodDeclaration();  } else if (cnt.symbol.equals("public") || cnt.symbol.equals("static") || cnt.symbol.equals("final")) {  constantDeclaration();  }  }  //Constructor  public void constructorDeclaration() throws IOException {  parameters();  throws1();  constructorBody();  }  public void parameters() throws IOException {  this.cnt.accept("(");  while (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  formalParameter();  }  this.cnt.accept(")");  }  public void formalParameter() throws IOException {  dataType();  identifier();  if(cnt.symbol.equals(",")){  this.cnt.accept(",");  }  }  public void throws1() throws IOException {  if (cnt.symbol.equals("throws")) {  this.cnt.accept("throws");  identifier();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  identifier();  }  }  }  public void constructorBody() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("this") || cnt.symbol.equals("super") || cnt.symbol.equals("new")  || cnt.symbol.equals("++") || cnt.symbol.equals("--") || cnt.symbol.equals("{")  || cnt.symbol.equals(";") || cnt.symbol.equals("switch") || cnt.symbol.equals("do")  || cnt.symbol.equals("break") || cnt.symbol.equals("continue") || cnt.symbol.equals("return")  || cnt.symbol.equals("synchronized") || cnt.symbol.equals("throws") || cnt.symbol.equals("try")  || cnt.symbol.equals("if") || cnt.symbol.equals("while") || cnt.symbol.equals("for")  || cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  blockStatement();  }  this.cnt.accept("}");  }  public void explicitConstructorInvocation() throws IOException {  switch (cnt.symbol) {  case ("this"):  this.cnt.accept("this");  this.cnt.accept("(");  argumentList();  this.cnt.accept(")");  this.cnt.accept(";");  break;  case ("super"):  this.cnt.accept("super");  this.cnt.accept("(");  argumentList();  this.cnt.accept(")");  this.cnt.accept(";");  break;  }  }  //Field  public void fieldDeclaration() throws IOException {  fieldAdditionalModifiers();  dataType();  variableDeclarators();  this.cnt.accept(";");  }  public void staticFieldDeclaration() throws IOException {  fieldModifier2Initializer();  dataType();  variableDeclarators();  this.cnt.accept(";");  }  public void fieldModifier2() throws IOException {  this.cnt.accept("transient");  }  public void fieldModifier3() throws IOException {  this.cnt.accept("volatile");  }  public void staticModifier() throws IOException {  this.cnt.accept("static");  }  public void fieldAdditionalModifiers() throws IOException {  switch (cnt.symbol) {  case ("transient"):  fieldModifier2();  staticModifierInitializer();  break;  case ("volatile"):  fieldModifier3();  fieldModifier3Declaration();  break;  }  }  public void staticModifierInitializer() throws IOException {  switch (cnt.symbol) {  case ("volatile"):  fieldModifier3();  staticModifier();  break;  case ("static"):  staticModifier();  fieldModifier3Initializer();  break;  }  }  public void fieldModifier2Initializer() throws IOException {  switch (cnt.symbol) {  case ("volatile"):  fieldModifier3();  fieldModifier2Option();  break;  case ("transient"):  fieldModifier2();  fieldModifier3Initializer();  break;  }  }    public void fieldModifier2Option() throws IOException {  if(cnt.symbol.equals("transient")){  this.cnt.accept("transient");  }  }  public void fieldModifier3Initializer() throws IOException {  if (cnt.symbol.equals("volatile")) {  fieldModifier3();  }  }  public void fieldModifier3Declaration() throws IOException {  fieldModifier2();  staticModifier();  }  public void variableOrMethodOption() throws IOException {  if (cnt.symbol.equals("(")) {  parameters();  throws1();  block();  } else if (cnt.symbol.equals("[") || cnt.symbol.equals("=")  || cnt.symbol.equals(",") || cnt.symbol.equals(";")) {  variableOption();  variableLooping();  this.cnt.accept(";");  }  }  public void variableDeclarators() throws IOException {  variableDeclarator();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  variableDeclarator();  }  }  public void variableDeclarator() throws IOException {  if (cnt.symbol.equals("\_") || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  variableOption();  } else if (cnt.symbol.equals("[")) {  arrayAfterDataType();  }  }  public void arrayAfterDataType() throws IOException {  this.cnt.accept("[");  this.cnt.accept("]");  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  arrayDeclaration();  }  public void arrayDeclaration() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  arrayInitializer();  }  }  public void variableOption() throws IOException {  switch (cnt.symbol) {  case ("="):  variableOperator();  break;  case ("["):  this.cnt.accept("[");  this.cnt.accept("]");  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  arrayInitializer();  break;  }  }    public void variableOption2() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("(") || cnt.symbol.equals("+")  || cnt.symbol.equals("-") || cnt.symbol.equals("~") || cnt.symbol.equals("new")  || cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("null") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 34)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 39) || (cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  variableInitializer();  } else if(cnt.symbol.equals("{")){  this.cnt.accept("{");  arrayTypeInitializer();  this.cnt.accept("}");  }  }    public void variableOperator() throws IOException {  if (cnt.symbol.equals("=")) {  this.cnt.accept("=");  variableOption2();  }  }  public void variableInitializers() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("(") || cnt.symbol.equals("+")  || cnt.symbol.equals("-") || cnt.symbol.equals("~") || cnt.symbol.equals("new")  || cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("null") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 34)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 39) || (cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  variableInitializer();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  variableInitializer();  }  }  }  public void variableInitializer() throws IOException {  expression();  }  public void variableLooping() throws IOException {  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  identifier();  variableOption();  }  }  public void arrayInitializer() throws IOException {  this.cnt.accept("=");  this.cnt.accept("{");  arrayTypeInitializer();  this.cnt.accept("}");  }  public void arrayTypeInitializer() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("(") || cnt.symbol.equals("+")  || cnt.symbol.equals("-") || cnt.symbol.equals("~") || cnt.symbol.equals("new")  || cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("null") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 34)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) == 39) || (cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  variableInitializers();  } else if (cnt.symbol.equals("{")) {  this.cnt.accept("{");  variableInitializers();  this.cnt.accept("}");  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  this.cnt.accept("{");  variableInitializers();  this.cnt.accept("}");  }  }  }  public void constantDeclaration() throws IOException {  constantModifiers();  dataType();  variableDeclarator();  this.cnt.accept(";");  }  public void constantModifiers() throws IOException {  switch (cnt.symbol) {  case ("public"):  this.cnt.accept("public");  break;  case ("static"):  this.cnt.accept("static");  break;  case ("final"):  this.cnt.accept("final");  break;  }  }  //Method  public void methodInitializer() throws IOException {  switch (cnt.symbol) {  case ("synchronized"):  case ("void"):  methodDeclaration();  block();  break;  case ("native"):  nativeMethodDeclaration();  break;  }  }  public void methodDeclaration() throws IOException {  switch (cnt.symbol) {  case ("synchronized"):  methodAdditionalModifier();  this.cnt.accept("void");  methodDeclarator();  throws1();  break;  case ("void"):  this.cnt.accept("void");  methodDeclarator();  throws1();  break;  }  }  public void staticMethodDeclaration() throws IOException {  switch (cnt.symbol) {  case ("final"):  staticAdditionalMod();  resultType();  methodDeclarator();  throws1();  break;  case ("void"):  this.cnt.accept("void");  methodDeclarator();  throws1();  break;  }  }  public void resultType() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  dataType();  } else if (cnt.symbol.equals("void")) {  this.cnt.accept("void");  }  }  public void finalModifier() throws IOException {  this.cnt.accept("final");  }  public void synchronizedModifier() throws IOException {  this.cnt.accept("synchronized");  }  public void methodAdditionalModifier() throws IOException {  synchronizedModifier();  synchronizedAdditionalMod();  }  public void staticAdditionalMod() throws IOException {  finalModifier();  synchronizedModInitializer();  }  public void synchronizedMethodDeclaration() throws IOException {  if (cnt.symbol.equals("final")) {  finalModInitializer();  }  }  public void finalAdditionalMod() throws IOException {  if (cnt.symbol.equals("synchronized")) {  synchronizedModifier();  staticModInitializer();  }  }  public void synchronizedAdditionalMod() throws IOException {  if (cnt.symbol.equals("static")) {  staticModifier();  finalModInitializer();  } else if (cnt.symbol.equals("final")) {  finalModifier();  staticModInitializer();  }  }  public void staticModInitializer() throws IOException {  if (cnt.symbol.equals("static")) {  staticModifier();  }  }  public void finalModInitializer() throws IOException {  if (cnt.symbol.equals("final")) {  finalModifier();  }  }  public void synchronizedModInitializer() throws IOException {  if (cnt.symbol.equals("synchronized")) {  synchronizedModifier();  }  }  public void methodDeclarator() throws IOException {  identifier();  parameters();  }  public void nativeMethodDeclaration() throws IOException {  nativeModifier();  resultType();  methodDeclarator();  throws1();  }  public void nativeModifier() throws IOException {  this.cnt.accept("native");  }  public void abstractMethodDeclaration() throws IOException {  abstractModifier();  resultType();  methodDeclarator();  throws1();  }  public void methodInvocation() throws IOException {  identifier();  this.cnt.accept("(");  argumentList();  this.cnt.accept(")");  }  //Data Type  public void dataType() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  dataPrimitive();  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  }  }  public void dataPrimitive() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  primitiveType();  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  }  }  public void primitiveType() throws IOException {  switch (cnt.symbol) {  case ("boolean"):  this.cnt.accept("boolean");  break;  case ("float"):  this.cnt.accept("float");  break;  case ("double"):  this.cnt.accept("double");  break;  case ("byte"):  this.cnt.accept("byte");  break;  case ("short"):  this.cnt.accept("short");  break;  case ("int"):  this.cnt.accept("int");  break;  case ("long"):  this.cnt.accept("long");  break;  case ("char"):  this.cnt.accept("char");  break;  }  }  //Statement  public void block() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals(";")  || cnt.symbol.equals("switch") || cnt.symbol.equals("do") || cnt.symbol.equals("break")  || cnt.symbol.equals("continue") || cnt.symbol.equals("return") || cnt.symbol.equals("synchronized")  || cnt.symbol.equals("throws") || cnt.symbol.equals("try") || cnt.symbol.equals("if")  || cnt.symbol.equals("while") || cnt.symbol.equals("for") || cnt.symbol.equals("super")  || cnt.symbol.equals("this") || cnt.symbol.equals("--") || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  blockStatement();  }  this.cnt.accept("}");  }  public void blockStatement() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  dataPrimitive();  variableDeclarators();  this.cnt.accept(";");  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  while(cnt.symbol.equals(".")){  this.cnt.accept(".");  identifier();  }  blockStatementOption();  } else if (cnt.symbol.equals("{") || cnt.symbol.equals(";")  || cnt.symbol.equals("switch") || cnt.symbol.equals("do") || cnt.symbol.equals("break")  || cnt.symbol.equals("continue") || cnt.symbol.equals("return") || cnt.symbol.equals("synchronized")  || cnt.symbol.equals("throws") || cnt.symbol.equals("try") || cnt.symbol.equals("if")  || cnt.symbol.equals("while") || cnt.symbol.equals("for")) {  statementWithoutExpressionStatement();  } else if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--") || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  expressionStatementWithoutIdentifier();  }  }  public void blockStatementOption() throws IOException {  if (cnt.symbol.equals("[")) {  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  variableDeclarators();  this.cnt.accept(";");  } else if (cnt.symbol.equals(":")) {  this.cnt.accept(":");  statement();  } else if (cnt.symbol.equals("--") || cnt.symbol.equals("++") || cnt.symbol.equals("(") || cnt.symbol.equals(".")  || cnt.symbol.equals("=") || cnt.symbol.equals("\*=") || cnt.symbol.equals("/=") || cnt.symbol.equals("%=")  || cnt.symbol.equals("+=") || cnt.symbol.equals("-=") || cnt.symbol.equals("<<=") || cnt.symbol.equals(">>=")  || cnt.symbol.equals(">>>=") || cnt.symbol.equals("&=") || cnt.symbol.equals("^=") || cnt.symbol.equals("|=")) {  postIdentifier();  this.cnt.accept(";");  }  }  public void statementWithoutExpressionStatement() throws IOException {  switch (cnt.symbol) {  case "{":  block();  break;  case ";":  emptyStatement();  break;  case "switch":  switchStatement();  break;  case "do":  doStatement();  break;  case "break":  breakStatement();  break;  case "continue":  continueStatement();  break;  case "return":  returnStatement();  break;  case "synchronized":  synchronizedStatement();  break;  case "throws":  throwsStatement();  break;  case "try":  tryStatement();  break;  case "if":  ifStatement();  break;  case "while":  whileStatement();  break;  case "for":  forStatement();  break;  }  }  public void statement() throws IOException {  if (cnt.symbol.equals("{") || cnt.symbol.equals(";")  || cnt.symbol.equals("switch") || cnt.symbol.equals("do") || cnt.symbol.equals("break")  || cnt.symbol.equals("continue") || cnt.symbol.equals("return") || cnt.symbol.equals("synchronized")  || cnt.symbol.equals("throws") || cnt.symbol.equals("try") || cnt.symbol.equals("if")  || cnt.symbol.equals("while") || cnt.symbol.equals("for")) {  statementWithoutExpressionStatement();  } else if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  expressionStatement();  }  }  public void emptyStatement() throws IOException {  this.cnt.accept(";");  }  public void labeledStatement() throws IOException {  identifier();  this.cnt.accept(":");  statement();  }  public void expressionStatementWithoutIdentifier() throws IOException {  if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  statementExpr();  this.cnt.accept(";");  }  }  public void expressionStatement() throws IOException {  if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  statementExpr();  this.cnt.accept(";");  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  identifierStatement();  }  }  public void identifierStatement() throws IOException {  if (cnt.symbol.equals("--") || cnt.symbol.equals("++") || cnt.symbol.equals("(") || cnt.symbol.equals(".")) {  postIdentifier();  this.cnt.accept(";");  } else if (cnt.symbol.equals(":")) {  this.cnt.accept(":");  statement();  }  }  public void statementExpression() throws IOException {  if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  statementExpr();  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  postIdentifier();  }  }  public void statementExpr() throws IOException {  switch (cnt.symbol) {  case "super":  this.cnt.accept("super");  thisSuperOption();  break;  case "this":  this.cnt.accept("this");  thisSuperOption();  break;  case "--":  case "++":  incrementDecrement();  identifier();  break;  case "new":  this.cnt.accept("new");  identifier();  classInstanceCreationExpression();  break;  }  }    public void thisSuperOption() throws IOException{  switch(cnt.symbol){  case ("."):  this.cnt.accept(".");  identifier();  while(cnt.symbol.equals(".")){  this.cnt.accept(".");  identifier();  }  arrayAccess();  assignment();  break;  case ("("):  parameters();  break;  }  }  public void postIdentifier() throws IOException {  switch (cnt.symbol) {  case "--":  case "++":  incrementDecrement();  break;  case "(":  this.cnt.accept("(");  argumentList();  this.cnt.accept(")");  break;  case ".":case "=":case "\*=":case "/=":case "%=":case"+=":  case "-=":case "<<=":case ">>=":case ">>>=":case "&=":  case "^=":case "|=":  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  identifier();  }  assignment();  break;  }  }  //Branching  public void ifStatement() throws IOException {  this.cnt.accept("if");  this.cnt.accept("(");  expression();  this.cnt.accept(")");  statement();  elseStatement();  }  public void elseStatement() throws IOException {  if (cnt.symbol.equals("else")) {  this.cnt.accept("else");  statement();  }  }  public void switchStatement() throws IOException {  this.cnt.accept("switch");  this.cnt.accept("(");  expression();  this.cnt.accept(")");  switchBlock();  }  public void switchBlock() throws IOException {  this.cnt.accept("{");  while (cnt.symbol.equals("case") || cnt.symbol.equals("default")) {  switchBlockStatementGroup();  }  this.cnt.accept("}");  }  public void switchBlockStatementGroup() throws IOException {  switchLabel();  while (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")    || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals(";")  || cnt.symbol.equals("switch") || cnt.symbol.equals("do") || cnt.symbol.equals("break")  || cnt.symbol.equals("continue") || cnt.symbol.equals("return") || cnt.symbol.equals("synchronized")  || cnt.symbol.equals("throws") || cnt.symbol.equals("try") || cnt.symbol.equals("if")  || cnt.symbol.equals("while") || cnt.symbol.equals("for") || cnt.symbol.equals("super")  || cnt.symbol.equals("this") || cnt.symbol.equals("--") || cnt.symbol.equals("++") || cnt.symbol.equals("new")) {  blockStatement();  }  }  public void switchLabel() throws IOException {  if (cnt.symbol.equals("case")) {  this.cnt.accept("case");  expression();  this.cnt.accept(":");  } else if (cnt.symbol.equals("default")) {  this.cnt.accept("default");  this.cnt.accept(":");  }  }  //Looping  public void whileStatement() throws IOException {  this.cnt.accept("while");  this.cnt.accept("(");  expression();  this.cnt.accept(")");  statement();  }  public void doStatement() throws IOException {  this.cnt.accept("do");  statement();  this.cnt.accept("while");  this.cnt.accept("(");  expression();  this.cnt.accept(")");  this.cnt.accept(";");  }  public void forStatement() throws IOException {  this.cnt.accept("for");  this.cnt.accept("(");  forInit();  this.cnt.accept(";");  expression();  this.cnt.accept(";");  forUpdate();  this.cnt.accept(")");  statement();  }  public void localVariableDeclaration() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  dataType();  variableDeclarators();  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("[")) {  variableDeclarators();  }  }  public void forInit() throws IOException {  if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  statementExpressionList();  } else if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char") || cnt.symbol.equals("\_")    || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("[")) {  localVariableDeclaration();  }  }  public void forUpdate() throws IOException {  if (cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("new") || cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  statementExpressionList();  }  }  public void statementExpressionList() throws IOException {  statementExpression();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  statementExpression();  }  }  public void breakStatement() throws IOException {  this.cnt.accept("break");  breakContinueIdentifier();  this.cnt.accept(";");  }  public void continueStatement() throws IOException {  this.cnt.accept("continue");  breakContinueIdentifier();  this.cnt.accept(";");  }  public void breakContinueIdentifier() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  }  }  public void returnStatement() throws IOException {  this.cnt.accept("return");  expression();  this.cnt.accept(";");  }  public void throwsStatement() throws IOException {  this.cnt.accept("throw");  expression();  this.cnt.accept(";");  }  public void synchronizedStatement() throws IOException {  this.cnt.accept("synchronized");  this.cnt.accept("(");  expression();  this.cnt.accept(")");  block();  }  public void tryStatement() throws IOException {  this.cnt.accept("try");  block();  catchStatement();  }  public void catchStatement() throws IOException {  if (cnt.symbol.equals("catch")) {  catches();  finallyInitializer();  } else if (cnt.symbol.equals("finally")) {  finallyStatement();  }  }  public void finallyInitializer() throws IOException {  if (cnt.symbol.equals("finally")) {  finallyStatement();  }  }  public void catches() throws IOException {  catchClause();  while (cnt.symbol.equals("catch")) {  catchClause();  }  }  public void catchClause() throws IOException {  this.cnt.accept("catch");  this.cnt.accept("(");  formalParameter();  this.cnt.accept(")");  block();  }  public void finallyStatement() throws IOException {  this.cnt.accept("finally");  block();  }  //Expression  public void expression() throws IOException {  additiveExpression();  multiplicativeOperator();  }  public void multiplicativeOperator() throws IOException {  if (cnt.symbol.equals("\*")) {  this.cnt.accept("\*");  expression();  } else if (cnt.symbol.equals("/")) {  this.cnt.accept("/");  expression();  } else if (cnt.symbol.equals("%")) {  this.cnt.accept("%");  expression();  }  }  public void additiveExpression() throws IOException {  shiftExpression();  additiveOperator();  }  public void additiveOperator() throws IOException {  if (cnt.symbol.equals("+")) {  this.cnt.accept("+");  additiveExpression();  } else if (cnt.symbol.equals("-")) {  this.cnt.accept("-");  additiveExpression();  }  }  public void shiftExpression() throws IOException {  relationalExpression();  shiftOperator();  }  public void shiftOperator() throws IOException {  if (cnt.symbol.equals(">>")) {  this.cnt.accept(">>");  shiftExpression();  } else if (cnt.symbol.equals("<<")) {  this.cnt.accept("<<");  shiftExpression();  } else if (cnt.symbol.equals(">>>")) {  this.cnt.accept(">>>");  shiftExpression();  }  }  public void relationalExpression() throws IOException {  equalityExpression();  relationalOperator();  }  public void relationalOperator() throws IOException {  if (cnt.symbol.equals("<")) {  this.cnt.accept("<");  relationalExpression();  } else if (cnt.symbol.equals(">")) {  this.cnt.accept(">");  relationalExpression();  } else if (cnt.symbol.equals("<=")) {  this.cnt.accept("<=");  relationalExpression();  } else if (cnt.symbol.equals(">=")) {  this.cnt.accept(">=");  relationalExpression();  } else if (cnt.symbol.equals("instanceof")) {  this.cnt.accept("instanceof");  relationalExpression();  }  }  public void equalityExpression() throws IOException {  andExpression();  equalityOperator();  }  public void equalityOperator() throws IOException {  if (cnt.symbol.equals("==")) {  this.cnt.accept("==");  equalityExpression();  } else if (cnt.symbol.equals("!=")) {  this.cnt.accept("!=");  equalityExpression();  }  }  public void andExpression() throws IOException {  exclusiveOr();  andOperator();  }  public void andOperator() throws IOException {  if (cnt.symbol.equals("&")) {  this.cnt.accept("&");  andExpression();  }  }  public void exclusiveOr() throws IOException {  inclusiveOr();  exclusiveOperator();  }  public void exclusiveOperator() throws IOException {  if (cnt.symbol.equals("^")) {  this.cnt.accept("^");  exclusiveOr();  }  }  public void inclusiveOr() throws IOException {  conditionalAnd();  inclusiveOrOperator();  }  public void inclusiveOrOperator() throws IOException {  if (cnt.symbol.equals("|")) {  this.cnt.accept("|");  inclusiveOr();  }  }  public void conditionalAnd() throws IOException {  conditionalOr();  conditionalAndOperator();  }  public void conditionalAndOperator() throws IOException {  if (cnt.symbol.equals("&&")) {  this.cnt.accept("&&");  conditionalAnd();  }  }  public void conditionalOr() throws IOException {  unaryExpression();  conditionalOrOperator();  }  public void conditionalOrOperator() throws IOException {  if (cnt.symbol.equals("?")) {  this.cnt.accept("?");  expression();  this.cnt.accept(":");  expression();  }  }  public void castType() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  primitiveType();  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  }  }  public void unaryExpression() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  expressionName();  postExpressionName();  } else if (cnt.symbol.equals("--") || cnt.symbol.equals("++")) {  incrementDecrement();  expressionName();  } else if (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("\'") || cnt.symbol.equals("\"") || cnt.symbol.equals("null")  || cnt.symbol.equals("new") || cnt.symbol.equals("super") || cnt.symbol.equals("this")) {  primary();  } else if (cnt.symbol.equals("(")) {  this.cnt.accept("(");  castType();  this.cnt.accept(")");  unaryExpression();  } else if (cnt.symbol.equals("+")) {  this.cnt.accept("+");  unaryExpression();  } else if (cnt.symbol.equals("-")) {  this.cnt.accept("-");  unaryExpression();  } else if (cnt.symbol.equals("~")) {  this.cnt.accept("~");  unaryExpression();  }  }  public void incrementDecrement() throws IOException {  if (cnt.symbol.equals("--")) {  this.cnt.accept("--");  } else if (cnt.symbol.equals("++")) {  this.cnt.accept("++");  }  }  public void postExpressionName() throws IOException {  if (cnt.symbol.equals("--") || cnt.symbol.equals("++")) {  incrementDecrement();  } else if (cnt.symbol.equals(".")) {  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  identifier();  }  arrayOrAssignment();  }  }  public void arrayOrAssignment() throws IOException {  if (cnt.symbol.equals("[")) {  arrayAccess();  } else if (cnt.symbol.equals("=") || cnt.symbol.equals("\*=") || cnt.symbol.equals("/=")  || cnt.symbol.equals("%=") || cnt.symbol.equals("+=") || cnt.symbol.equals("-=")  || cnt.symbol.equals("<<=") || cnt.symbol.equals(">>=") || cnt.symbol.equals(">>>=")  || cnt.symbol.equals("&=") || cnt.symbol.equals("^=") || cnt.symbol.equals("|=")) {  assignmentOperator();  expression();  }  }  public void primary() throws IOException {  if (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("\'") || cnt.symbol.equals("\"") || cnt.symbol.equals("null")) {  literal();  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  methodInvocation();  }  } else if (cnt.symbol.equals("new")) {  this.cnt.accept("new");  instanceCreationExpression();  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  methodInvocation();  }  } else if (cnt.symbol.equals("super") || cnt.symbol.equals("this")) {  fieldAccess();  assignment();  }  }  public void instanceCreationExpression() throws IOException {  if (cnt.symbol.equals("boolean") || cnt.symbol.equals("float") || cnt.symbol.equals("double")  || cnt.symbol.equals("byte") || cnt.symbol.equals("short") || cnt.symbol.equals("int")  || cnt.symbol.equals("long") || cnt.symbol.equals("char")) {  primitiveType();  dimExpressions();  dims();  } else if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  identifier();  creationExpression();  }  }  public void creationExpression() throws IOException {  if (cnt.symbol.equals("(")) {  classInstanceCreationExpression();  } else if (cnt.symbol.equals("[")) {  arrayCreation();  }  }  public void classInstanceCreationExpression() throws IOException {  this.cnt.accept("(");  argumentList();  this.cnt.accept(")");  }  public void argumentList() throws IOException {  if (cnt.symbol.equals("\_")  || (cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || cnt.symbol.equals("--")  || cnt.symbol.equals("++") || cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9") || cnt.symbol.equals("true")  || cnt.symbol.equals("false") || cnt.symbol.equals("\'") || cnt.symbol.equals("\"") || cnt.symbol.equals("null")  || cnt.symbol.equals("new") || cnt.symbol.equals("super") || cnt.symbol.equals("this") || cnt.symbol.equals("(")  || cnt.symbol.equals("+") || cnt.symbol.equals("-") || cnt.symbol.equals("~")) {  expression();  while (cnt.symbol.equals(",")) {  this.cnt.accept(",");  expression();  }  }  }  public void arrayCreation() throws IOException {  dimExpressions();  dims();  }  public void dimExpressions() throws IOException {  dimExpression();  while (cnt.symbol.equals("[")) {  dimExpression();  }  }  public void dimExpression() throws IOException {  this.cnt.accept("[");  expression();  this.cnt.accept("]");  }  public void dims() throws IOException {  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  this.cnt.accept("]");  }  }  public void assignment() throws IOException {  if (cnt.symbol.equals("=") || cnt.symbol.equals("\*=") || cnt.symbol.equals("/=")  || cnt.symbol.equals("%=") || cnt.symbol.equals("+=") || cnt.symbol.equals("-=")  || cnt.symbol.equals("<<=") || cnt.symbol.equals(">>=") || cnt.symbol.equals(">>>=")  || cnt.symbol.equals("&=") || cnt.symbol.equals("^=") || cnt.symbol.equals("|=")) {  assignmentOperator();  expression();  }  }  public void assignmentOperator() throws IOException {  if (cnt.symbol.equals("=")) {  this.cnt.accept("=");  } else if (cnt.symbol.equals("\*=")) {  this.cnt.accept("\*=");  } else if (cnt.symbol.equals("/=")) {  this.cnt.accept("/=");  } else if (cnt.symbol.equals("%=")) {  this.cnt.accept("%=");  } else if (cnt.symbol.equals("+=")) {  this.cnt.accept("+=");  } else if (cnt.symbol.equals("-=")) {  this.cnt.accept("-=");  } else if (cnt.symbol.equals("<<=")) {  this.cnt.accept("<<=");  } else if (cnt.symbol.equals(">>=")) {  this.cnt.accept(">>=");  } else if (cnt.symbol.equals(">>>=")) {  this.cnt.accept(">>>=");  } else if (cnt.symbol.equals("&=")) {  this.cnt.accept("&=");  } else if (cnt.symbol.equals("^=")) {  this.cnt.accept("^=");  } else if (cnt.symbol.equals("|=")) {  this.cnt.accept("|=");  }  }  public void fieldAccess() throws IOException {  if (cnt.symbol.equals("super")) {  this.cnt.accept("super");  this.cnt.accept(".");  identifier();  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  identifier();  }  arrayAccess();  } else if (cnt.symbol.equals("this")) {  this.cnt.accept("this");  this.cnt.accept(".");  identifier();  while (cnt.symbol.equals(".")) {  this.cnt.accept(".");  identifier();  }  arrayAccess();  }  }  public void arrayAccess() throws IOException {  while (cnt.symbol.equals("[")) {  this.cnt.accept("[");  expression();  this.cnt.accept("]");  }  }  public void expressionName() throws IOException {  identifier();  }  public void identifier() throws IOException {  if ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  alphabet();  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || (cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57)) || cnt.symbol.equals("\_")) {  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  digit();  }  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  alphabet();  }  while (cnt.symbol.equals("\_")) {  this.cnt.accept("\_");  }  }  } else if (cnt.symbol.equals("\_")) {  this.cnt.accept("\_");  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122)) || (cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57)) || cnt.symbol.equals("\_")) {  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  digit();  }  while ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  alphabet();  }  while (cnt.symbol.equals("\_")) {  this.cnt.accept("\_");  }  }  }  }  //Literal  public void literal() throws IOException {  if (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  numberLiteral();  } else if (cnt.symbol.equals("true") || cnt.symbol.equals("false")) {  booleanLiteral();  } else if (cnt.symbol.equals("\'")) {  characterLiteral();  } else if (cnt.symbol.equals("\"")) {  stringLiteral();  } else if (cnt.symbol.equals("null")) {  nullLiteral();  }  }  public void numberLiteral() throws IOException {  if (cnt.symbol.equals("0")) {  this.cnt.accept("0");  zeroNumberOption();  integerTypeSuffix();  } else if (cnt.symbol.equals("1") || cnt.symbol.equals("2") || cnt.symbol.equals("3")  || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  nonZeroDigit();  while (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  digit();  }  nonZeroOption();  }  }  public void zeroNumberOption() throws IOException {  if (cnt.symbol.equals("x") || cnt.symbol.equals("X")) {  hexNumeral();  while (cnt.symbol.equals("x") || cnt.symbol.equals("X")) {  hexNumeral();  }  integerTypeSuffix();  } else if (cnt.symbol.equals("0")) {  octalNumeral();  while (cnt.symbol.equals("0")) {  octalNumeral();  }  integerTypeSuffix();  } else if (cnt.symbol.equals(".")) {  floatingPointLiteral();  }  }  public void nonZeroOption() throws IOException {  if (cnt.symbol.equals("l") || cnt.symbol.equals("L")) {  integerTypeSuffix();  } else if (cnt.symbol.equals(".")) {  floatingPointLiteral();  }  }  public void integerTypeSuffix() throws IOException {  if (cnt.symbol.equals("l")) {  this.cnt.accept("l");  } else if (cnt.symbol.equals("L")) {  this.cnt.accept("L");  }  }  public void digit() throws IOException {  if (cnt.symbol.equals("0")) {  this.cnt.accept("0");  } else if (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 49 && (int) cnt.symbol.charAt(0) <= 57) {  nonZeroDigit();  }  }  public void nonZeroDigit() throws IOException {  switch (cnt.symbol) {  case "1":  this.cnt.accept("1");  break;  case "2":  this.cnt.accept("2");  break;  case "3":  this.cnt.accept("3");  break;  case "4":  this.cnt.accept("4");  break;  case "5":  this.cnt.accept("5");  break;  case "6":  this.cnt.accept("6");  break;  case "7":  this.cnt.accept("7");  break;  case "8":  this.cnt.accept("8");  break;  case "9":  this.cnt.accept("9");  break;  }  }  public void hexNumeral() throws IOException {  if (cnt.symbol.equals("x")) {  this.cnt.accept("x");  hexDigit();  } else if (cnt.symbol.equals("X")) {  this.cnt.accept("X");  hexDigit();  }  }  public void hexDigit() throws IOException {  switch (cnt.symbol) {  case "0":  this.cnt.accept("0");  break;  case "1":  this.cnt.accept("1");  break;  case "2":  this.cnt.accept("2");  break;  case "3":  this.cnt.accept("3");  break;  case "4":  this.cnt.accept("4");  break;  case "5":  this.cnt.accept("5");  break;  case "6":  this.cnt.accept("6");  break;  case "7":  this.cnt.accept("7");  break;  case "8":  this.cnt.accept("8");  break;  case "9":  this.cnt.accept("9");  break;  case "a":  this.cnt.accept("a");  break;  case "b":  this.cnt.accept("b");  break;  case "c":  this.cnt.accept("c");  break;  case "d":  this.cnt.accept("d");  break;  case "e":  this.cnt.accept("e");  break;  case "f":  this.cnt.accept("f");  break;  case "A":  this.cnt.accept("A");  break;  case "B":  this.cnt.accept("B");  break;  case "C":  this.cnt.accept("C");  break;  case "D":  this.cnt.accept("D");  break;  case "E":  this.cnt.accept("E");  break;  case "F":  this.cnt.accept("F");  break;  }  }  public void octalNumeral() throws IOException {  this.cnt.accept("0");  octalDigit();  }  public void octalDigit() throws IOException {  switch (cnt.symbol) {  case "0":  this.cnt.accept("0");  break;  case "1":  this.cnt.accept("1");  break;  case "2":  this.cnt.accept("2");  break;  case "3":  this.cnt.accept("3");  break;  case "4":  this.cnt.accept("4");  break;  case "5":  this.cnt.accept("5");  break;  case "6":  this.cnt.accept("6");  break;  case "7":  this.cnt.accept("7");  break;  }  }  public void floatingPointLiteral() throws IOException {  this.cnt.accept(".");  while (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  digit();  }  exponentPart();  floatTypeSuffix();  digit();  while (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  digit();  }  exponentPart();  floatTypeSuffix();  }  public void exponentPart() throws IOException {  if (cnt.symbol.equals("e") || cnt.symbol.equals("E")) {  exponentIndicator();  signedInteger();  }  }  public void exponentIndicator() throws IOException {  if (cnt.symbol.equals("e")) {  this.cnt.accept("e");  } else if (cnt.symbol.equals("E")) {  this.cnt.accept("E");  }  }  public void signedInteger() throws IOException {  sign();  digit();  while (cnt.symbol.equals("0") || cnt.symbol.equals("1") || cnt.symbol.equals("2")  || cnt.symbol.equals("3") || cnt.symbol.equals("4") || cnt.symbol.equals("5") || cnt.symbol.equals("6")  || cnt.symbol.equals("7") || cnt.symbol.equals("8") || cnt.symbol.equals("9")) {  digit();  }  }  public void sign() throws IOException {  if (cnt.symbol.equals("+")) {  this.cnt.accept("+");  } else if (cnt.symbol.equals("-")) {  this.cnt.accept("-");  }  }  public void floatTypeSuffix() throws IOException {  if (cnt.symbol.equals("f")) {  this.cnt.accept("f");  } else if (cnt.symbol.equals("F")) {  this.cnt.accept("F");  } else if (cnt.symbol.equals("d")) {  this.cnt.accept("d");  } else if (cnt.symbol.equals("D")) {  this.cnt.accept("D");  }  }  public void booleanLiteral() throws IOException {  if (cnt.symbol.equals("true")) {  this.cnt.accept("true");  } else if (cnt.symbol.equals("false")) {  this.cnt.accept("false");  }  }  public void characterLiteral() throws IOException {  if (cnt.symbol.equals("\'")) {  this.cnt.accept("\'");  characterLiteralOption();  this.cnt.accept("\'");  }  }  public void characterLiteralOption() throws IOException {  if ((cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 33 && (int) cnt.symbol.charAt(0) <= 38)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 40 && (int) cnt.symbol.charAt(0) <= 91)) {  singleCharacter();  } else if (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 93 && (int) cnt.symbol.charAt(0) <= 126) {  singleCharacter();  } else if (cnt.symbol.equals("\\t") || cnt.symbol.equals("\\b") || cnt.symbol.equals("\\n")  || cnt.symbol.equals("\\r") || cnt.symbol.equals("\\f") || cnt.symbol.equals("\\\'")  || cnt.symbol.equals("\\\"") || cnt.symbol.equals("\\\\")) {  escapeCharacter();  }  }  public void singleCharacter() throws IOException {  if (cnt.symbol.equals("!") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 35 && (int) cnt.symbol.charAt(0) <= 38)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 40 && (int) cnt.symbol.charAt(0) <= 91)) {  inputCharacter();  } else if (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 93 && (int) cnt.symbol.charAt(0) <= 126) {  inputCharacter();  } else if (cnt.symbol.equals("\"")) {  this.cnt.accept("\"");  }  }  public void stringLiteral() throws IOException {  this.cnt.accept("\"");  while (cnt.symbol.equals("!") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 35 && (int) cnt.symbol.charAt(0) <= 91)  || cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 93 && (int) cnt.symbol.charAt(0) <= 126  || cnt.symbol.equals("\'") || cnt.symbol.equals("\\t") || cnt.symbol.equals("\\b")  || cnt.symbol.equals("\\n") || cnt.symbol.equals("\\r") || cnt.symbol.equals("\\f")  || cnt.symbol.equals("\\\'") || cnt.symbol.equals("\\\"") || cnt.symbol.equals("\\\\")) {  stringCharacter();  }  this.cnt.accept("\"");  }  public void alphabet() throws IOException {  switch (cnt.symbol) {  case "A":  this.cnt.accept("A");  break;  case "B":  this.cnt.accept("B");  break;  case "C":  this.cnt.accept("C");  break;  case "D":  this.cnt.accept("D");  break;  case "E":  this.cnt.accept("E");  break;  case "F":  this.cnt.accept("F");  break;  case "G":  this.cnt.accept("G");  break;  case "H":  this.cnt.accept("H");  break;  case "I":  this.cnt.accept("I");  break;  case "J":  this.cnt.accept("J");  break;  case "K":  this.cnt.accept("K");  break;  case "L":  this.cnt.accept("L");  break;  case "M":  this.cnt.accept("M");  break;  case "N":  this.cnt.accept("N");  break;  case "O":  this.cnt.accept("O");  break;  case "P":  this.cnt.accept("P");  break;  case "Q":  this.cnt.accept("Q");  break;  case "R":  this.cnt.accept("R");  break;  case "S":  this.cnt.accept("S");  break;  case "T":  this.cnt.accept("T");  break;  case "U":  this.cnt.accept("U");  break;  case "V":  this.cnt.accept("V");  break;  case "W":  this.cnt.accept("W");  break;  case "X":  this.cnt.accept("X");  break;  case "Y":  this.cnt.accept("Y");  break;  case "Z":  this.cnt.accept("Z");  break;  case "a":  this.cnt.accept("a");  break;  case "b":  this.cnt.accept("b");  break;  case "c":  this.cnt.accept("c");  break;  case "d":  this.cnt.accept("d");  break;  case "e":  this.cnt.accept("e");  break;  case "f":  this.cnt.accept("f");  break;  case "g":  this.cnt.accept("g");  break;  case "h":  this.cnt.accept("h");  break;  case "i":  this.cnt.accept("i");  break;  case "j":  this.cnt.accept("j");  break;  case "k":  this.cnt.accept("k");  break;  case "l":  this.cnt.accept("l");  break;  case "m":  this.cnt.accept("m");  break;  case "n":  this.cnt.accept("n");  break;  case "o":  this.cnt.accept("o");  break;  case "p":  this.cnt.accept("p");  break;  case "q":  this.cnt.accept("q");  break;  case "r":  this.cnt.accept("r");  break;  case "s":  this.cnt.accept("s");  break;  case "t":  this.cnt.accept("t");  break;  case "u":  this.cnt.accept("u");  break;  case "v":  this.cnt.accept("v");  break;  case "w":  this.cnt.accept("w");  break;  case "x":  this.cnt.accept("x");  break;  case "y":  this.cnt.accept("y");  break;  case "z":  this.cnt.accept("z");  break;  }  }  public void stringCharacter() throws IOException {  if (cnt.symbol.equals("!") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 35 && (int) cnt.symbol.charAt(0) <= 38)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 40 && (int) cnt.symbol.charAt(0) <= 91)) {  inputCharacter();  } else if (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 93 && (int) cnt.symbol.charAt(0) <= 126) {  inputCharacter();  } else if (cnt.symbol.equals("\'")) {  this.cnt.accept("\'");  } else if (cnt.symbol.equals("\\t") || cnt.symbol.equals("\\b") || cnt.symbol.equals("\\n")  || cnt.symbol.equals("\\r") || cnt.symbol.equals("\\f") || cnt.symbol.equals("\\\'")  || cnt.symbol.equals("\\\"") || cnt.symbol.equals("\\\\")) {  escapeCharacter();  }  }  public void inputCharacter() throws IOException {  if ((cnt.symbol.length() == 1 && ((int) cnt.symbol.toLowerCase().charAt(0) >= 97 && (int) cnt.symbol.toLowerCase().charAt(0) <= 122))) {  alphabet();  } else if ((cnt.symbol.length() == 1 && ((int) cnt.symbol.charAt(0) >= 48 && (int) cnt.symbol.charAt(0) <= 57))) {  digit();  } else if (cnt.symbol.equals("!")) {  this.cnt.accept("!");  } else if (cnt.symbol.equals("#")) {  this.cnt.accept("#");  } else if (cnt.symbol.equals("$")) {  this.cnt.accept("$");  } else if (cnt.symbol.equals("%")) {  this.cnt.accept("%");  } else if (cnt.symbol.equals("&")) {  this.cnt.accept("&");  } else if (cnt.symbol.equals("(")) {  this.cnt.accept("(");  } else if (cnt.symbol.equals(")")) {  this.cnt.accept(")");  } else if (cnt.symbol.equals("\*")) {  this.cnt.accept("\*");  } else if (cnt.symbol.equals("+")) {  this.cnt.accept("+");  } else if (cnt.symbol.equals(",")) {  this.cnt.accept(",");  } else if (cnt.symbol.equals("-")) {  this.cnt.accept("-");  } else if (cnt.symbol.equals(".")) {  this.cnt.accept(".");  } else if (cnt.symbol.equals("/")) {  this.cnt.accept("/");  } else if (cnt.symbol.equals(":")) {  this.cnt.accept(":");  } else if (cnt.symbol.equals(";")) {  this.cnt.accept(";");  } else if (cnt.symbol.equals("<")) {  this.cnt.accept("<");  } else if (cnt.symbol.equals("=")) {  this.cnt.accept("=");  } else if (cnt.symbol.equals(">")) {  this.cnt.accept(">");  } else if (cnt.symbol.equals("?")) {  this.cnt.accept("?");  } else if (cnt.symbol.equals("@")) {  this.cnt.accept("@");  } else if (cnt.symbol.equals("[")) {  this.cnt.accept("[");  } else if (cnt.symbol.equals("]")) {  this.cnt.accept("]");  } else if (cnt.symbol.equals("^")) {  this.cnt.accept("^");  } else if (cnt.symbol.equals("\_")) {  this.cnt.accept("\_");  } else if (cnt.symbol.equals("`")) {  this.cnt.accept("`");  } else if (cnt.symbol.equals("{")) {  this.cnt.accept("{");  } else if (cnt.symbol.equals("|")) {  this.cnt.accept("|");  } else if (cnt.symbol.equals("}")) {  this.cnt.accept("}");  } else if (cnt.symbol.equals("~")) {  this.cnt.accept("~");  }  }  public void allInputCharacter() throws IOException {  if (cnt.symbol.equals("!") || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 35 && (int) cnt.symbol.charAt(0) <= 38)  || (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 40 && (int) cnt.symbol.charAt(0) <= 91)) {  inputCharacter();  } else if (cnt.symbol.length() == 1 && (int) cnt.symbol.charAt(0) >= 93 && (int) cnt.symbol.charAt(0) <= 126) {  inputCharacter();  } else if (cnt.symbol.equals("\"")) {  this.cnt.accept("\"");  } else if (cnt.symbol.equals("\'")) {  this.cnt.accept("\'");  } else if (cnt.symbol.equals("\\")) {  this.cnt.accept("\\");  }  }  public void escapeCharacter() throws IOException {  if (cnt.symbol.equals("\\t")) {  this.cnt.accept("\\t");  } else if (cnt.symbol.equals("\\b")) {  this.cnt.accept("\\b");  } else if (cnt.symbol.equals("\\n")) {  this.cnt.accept("\\n");  } else if (cnt.symbol.equals("\\r")) {  this.cnt.accept("\\r");  } else if (cnt.symbol.equals("\\f")) {  this.cnt.accept("\\f");  } else if (cnt.symbol.equals("\\\'")) {  this.cnt.accept("\\\'");  } else if (cnt.symbol.equals("\\\"")) {  this.cnt.accept("\\\"");  } else if (cnt.symbol.equals("\\\\")) {  this.cnt.accept("\\\\");  }  }  public void nullLiteral() throws IOException {  this.cnt.accept("null");  }  } |

**Kelas Controller.java**

|  |
| --- |
| package Controller;  import Model.ProgramDeclaration;  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.File;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  /\*\*  \* @author Sukamto 23518017 Andreas Novian 23518002  \*/  public class Controller {  List<String> listReservedWord, listAlphanumeric, listSingleCharacters, listMultiCharacters;  BufferedReader br;  BufferedWriter bw;  String isiFile = ""; //isi file murni, belum diapa-apain  String[] listOfSymbols; //kumpulan simbol setelah dipisahin  public String symbol; //simbol yang sedang diproses  int cursor; //penunjuk simbol yang sedang dicek  boolean isError = false;  String namaFileOutput;  public Controller(String namaFileInput, String namaFileOutput) throws FileNotFoundException, IOException {  initList();  br = new BufferedReader(new FileReader(namaFileInput));  this.namaFileOutput = namaFileOutput;  }  private void initList() throws FileNotFoundException, IOException {  //simpan isi file reservedWord ke list  br = new BufferedReader(new FileReader("reservedWord.txt"));  listReservedWord = new ArrayList<>();  String currentLine;  while ((currentLine = br.readLine()) != null) {  listReservedWord.add(currentLine);  }  //simpan isi file alphanumeric ke list  br = new BufferedReader(new FileReader("alphanumeric.txt"));  listAlphanumeric = new ArrayList<>();  while ((currentLine = br.readLine()) != null) {  listAlphanumeric.add(currentLine);  }  //simpan isi file singleCharacters ke list  br = new BufferedReader(new FileReader("singleCharacters.txt"));  listSingleCharacters = new ArrayList<>();  while ((currentLine = br.readLine()) != null) {  listSingleCharacters.add(currentLine);  }  //simpan isi file multiCharacters ke list  br = new BufferedReader(new FileReader("multiCharacters.txt"));  listMultiCharacters = new ArrayList<>();  while ((currentLine = br.readLine()) != null) {  listMultiCharacters.add(currentLine);  }  }  public void start() throws IOException {  bw = new BufferedWriter(new FileWriter(new File(this.namaFileOutput)));  cursor = 0;  String currentLine;  while ((currentLine = br.readLine()) != null) {  isiFile += currentLine + "\n";  }  br.close();  listOfSymbols = parseSymbols(isiFile);  symbol = listOfSymbols[0];  new ProgramDeclaration(this).compilationUnit();  //cek apakah masih ada sisa input setelah program berakhir  //jika ada, tampilkan error dan print seluruh sisa file  if (cursor < listOfSymbols.length) {  bw.write("(Error)");  for (int i = cursor; i < listOfSymbols.length; i++) {  bw.write(listOfSymbols[i]);  }  isError = true;  }  if (!isError) {  bw.write("\nTidak ada error\n");  }  bw.close();  br.close();  }  public void accept(String terminal) throws IOException {  boolean isAccepted = false;  //akan di readNextSymbol terus selama belum di accept  //agar tidak mengacaukan sisa file yang tidak error  //misalnya: input = (x+x+).  //input yang bisa diterima adalah (x+x).  //maka outputnya adalah: (x+x(Error)+). sisa file ). tidak error  while (!isAccepted) {  System.out.println("terminal - symbol = " + terminal + " - " + symbol);  if (cursor >= listOfSymbols.length) {  isAccepted = true;  } else {  if (terminal.equals(symbol)) {  isAccepted = true;  bw.write(symbol);  } else {  isError = true;  bw.write("(Error)" + symbol);  }  if (symbol.equals(";") || symbol.equals("{") || symbol.equals("}")) {  bw.write("\n");  } else if (symbol.length() > 1 && !symbol.equals("this") && !symbol.equals("super")) {  bw.write(" ");  }  }  readNextSymbol();  }  }  private void readNextSymbol() {  cursor++;  if (cursor < listOfSymbols.length) {  this.symbol = listOfSymbols[cursor];  }  }  public String getSymbol() {  return this.symbol;  }  private String[] parseSymbols(String isiFile) throws IOException {  String temp;  String lastKnown = "";  isiFile = isiFile.replaceAll("\\s+", "");  List<String> result = new ArrayList<>();  for (int i = 0; i < isiFile.length(); i++) {  temp = isiFile.charAt(i) + "";  if (isReservedWord(lastKnown + temp) || isMultiCharacters(lastKnown + temp)) {  for (int j = 0; j < lastKnown.length(); j++) {  //result.remove("" + lastKnown.charAt(j));  result.remove(result.size() - 1);  }  result.add(lastKnown + temp);  lastKnown = "";  } else if (result.size() > 0 && isMultiCharacters(result.get(result.size() - 1) + temp) || result.size() > 0 && isReservedWord(result.get(result.size() - 1) + temp)) {  result.add(result.remove(result.size() - 1) + temp);  lastKnown = "";  } else {  result.add(temp);  if (lastKnown.equalsIgnoreCase("")) {  lastKnown = temp;  } else {  if (isSingleCharacters(temp) && isSingleCharacters("" + lastKnown.charAt(lastKnown.length() - 1))) {  lastKnown += temp;  } else if (isAlphanumeric(temp) && isAlphanumeric("" + lastKnown.charAt(lastKnown.length() - 1))) {  lastKnown += temp;  } else {  lastKnown = temp;  }  }  }  }  String[] arrResult = new String[result.size()];  for (int i = 0; i < result.size(); i++) {  arrResult[i] = result.get(i);  }  return arrResult;  }  private boolean isReservedWord(String in) {  for (String currentLine : listReservedWord) {  if (currentLine.equalsIgnoreCase(in)) {  return true;  }  }  return false;  }  private boolean isAlphanumeric(String in) {  for (String currentLine : listAlphanumeric) {  if (currentLine.equalsIgnoreCase(in)) {  return true;  }  }  return false;  }  private boolean isSingleCharacters(String in) {  for (String currentLine : listSingleCharacters) {  if (currentLine.equalsIgnoreCase(in)) {  return true;  }  }  return false;  }  private boolean isMultiCharacters(String in) {  for (String currentLine : listMultiCharacters) {  if (currentLine.equalsIgnoreCase(in)) {  return true;  }  }  return false;  }  } |

**Kelas Tester.java**

|  |
| --- |
| import Controller.Controller;  import java.io.IOException;  /\*\*  \* @author Sukamto 23518017 Andreas Novian 23518002  \*/  public class Tester {  public static void main(String[] args) throws IOException {  Controller ct = new Controller("input1.txt","output1.txt");  ct.start();  ct = new Controller("input2.txt","output2.txt");  ct.start();  ct = new Controller("input3.txt","output3.txt");  ct.start();  }  } |