Bringing Ownership Domains to Mainstream Java

Marwan Abi-Antoun Jonathan Aldrich

Institute for Software Research

Carnegie Mellon University

OOPSLA 2006

ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications

October 22-26, Portland, Oregon, USA

Can you find the bug?

Hint: “The big lie of object-oriented programming is that objects provide encapsulation” (Hogg)

```
class JavaClass {
   private List signers;

   public List getSigners() {
          return this.signers;
   }
}
// (Malicious) clients can mutate signers field!
class MaliciousClient extends ... {
   public void addTrojanHorse(JavaClass c)
   {
          List signers = c.getSigners();
          signers.add( this );
   }
}
```

Aliasing and failure of encapsulation

- Aliasing cannot be eliminated
  - Object-oriented design patterns rely on it
  - Can be controlled with language support
- Several solutions proposed
- AliasJava: Ownership Domains
  - Open-source compiler available
  - Language extension to Java
  - Barat infrastructure
  - Basic tool support

Ownership domains

- Each object defines groups (*ownership domains)* to hold its private state;
- Ownership domains useful to:
  - Separate internals of object from users of object
  - Ensure private state not leaked
  - Distinguish different “subsystems” within an object
- Ownership domains control aliasing:
  - Within a domain, there *can be* aliasing
  - *No* aliasing between two given domains
  - *Explicit* permissions for cross-domain access

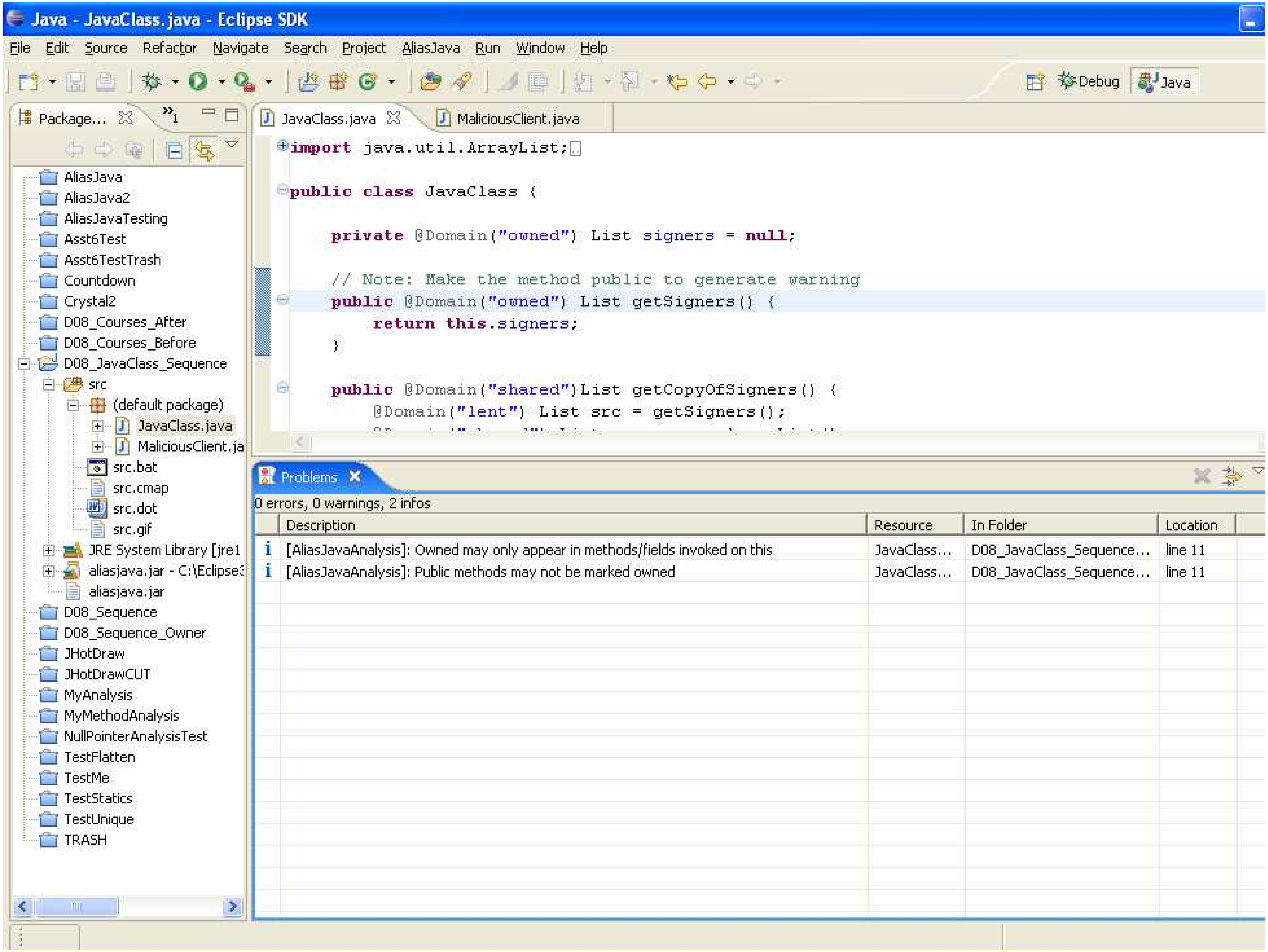
AliasJava with annotations

- Use Java 1.5 annotations (JSR 175)
- Move to Eclipse JDT infrastructure
- Advantages of using annotations
  - Improved tool support
  - Incrementally/partially annotate large codebase
  - Easier to add features to language
- Goal: improve usability and adoptability
  - No errors about inconsistent annotations
  - Add-on tool to supply reasonable defaults

Demo: Signers with AliasJava

```
public class JavaClass {

  private @Domain("owned") List signers = null;

  private @Domain("owned") List getSigners() {
   return this.signers;
  }

  public @Domain("shared") List getCopyOfSigners() {
   @Domain("lent")List src = getSigners();
   @Domain("shared") List copy = new ArrayList();
   // Copy array...
   return copy;
  }

   void setSigners(@Domain("owned")List signers){
    this.signers = signers;
   }
}
```

- Compile error for having **owned** private domain in public method
- `getSigners()` must be made private
- Clients can only invoke `getCopyOfSigners()`



```
import java.util.ArrayList;

public class JavaClass {

    private @Domain("owned") List signers = null;

    // Note: Make the method public to generate warning
    public @Domain("owned") List getSigners() {
        return this.signers;
    }

    public @Domain("shared")List getCopyOfSigners() {
        @Domain("lent") List src = getSigners();
```

0 errors, 0 warnings, 2 infos

	Description	Resource	In Folder	Location
i	[AliasJavaAnalysis]: Owned may only appear in methods/fields invoked on this	JavaClass...	D08_JavaClass_Sequence...	line 11
i	[AliasJavaAnalysis]: Public methods may not be marked owned	JavaClass...	D08_JavaClass_Sequence...	line 11

Annotation language

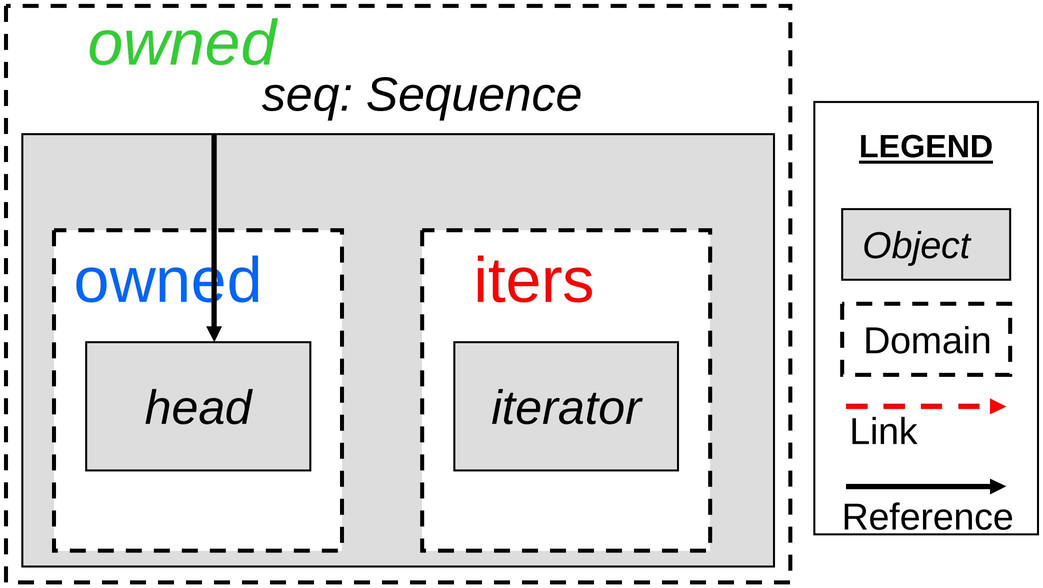
- `@Domains`: declare domains
- `@DomainParams`: declare *formal* domain parameters
- `@DomainLinks`: declare domain link specifications
- `@DomainInherits`: specify parameters for supertypes
- `@DomainReceiver`: specify annotation on receiver
- `@Domain`: specify object annotation, *actual* domain parameters and (optionally) array parameters *“annotation<domParam, …> [arrayParam, …]”*
- Annotation:
  - Special: “lent”, “unique”, “owned”, “shared”
  - Common: “iters” or “obj.iters”

Special alias types

- **owned**: instance confined within object (default domain)
- **unique**: instance passed linearly from one object to another
- **lent**: temporary alias within method
- **shared**: shared persistently or globally

Public, private ownership domains

```
@Domains({“iters”})
class Sequence {
 @Domain(“owned”)Cons head;

 public @Domain(“iters”)
 Iterator getIter() {
    return new Iterator(head);
  }
}
```



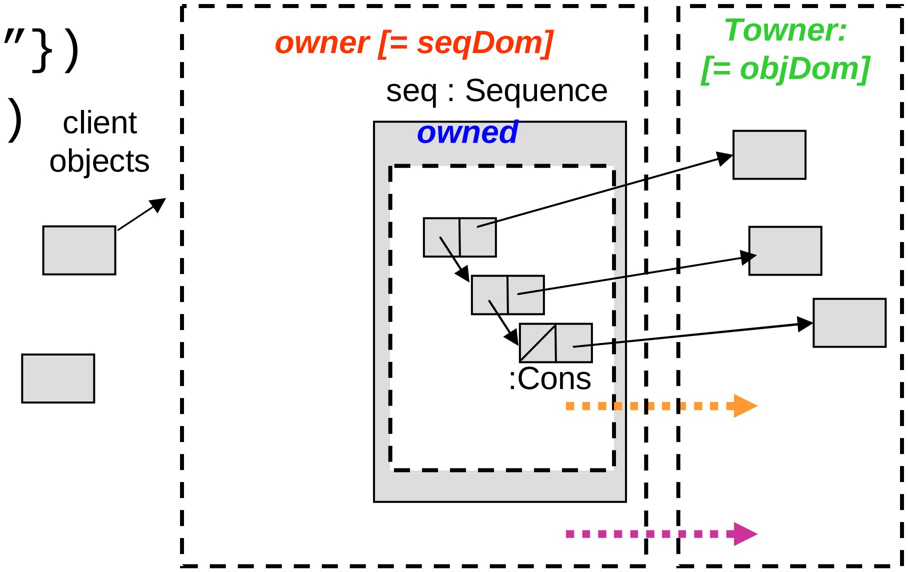
```
@Domain(“owned”)Sequence seq = new Sequence();
```

- Every object is in exactly one domain
- E.g., list in domain owned; iterators in domain iters
- Every object can have one or more domains
- E.g., domains owned and iters declared in Sequence

Ownership domain parameters

```
@DomainParams({“Towner”})
@DomainAssumes({“owner -> Towner”})
@DomainLinks({“owned -> Towner”})
class Sequence {
 @Domain(“owned<Towner>”)
 Cons head;
…
}
@DomainParams({“Towner”})
class Cons {
  @Domain(“Towner”)Object obj;
  @Domain(“owner<Towner>”)Cons next;
}
```

Cons.owner == Sequence.owned

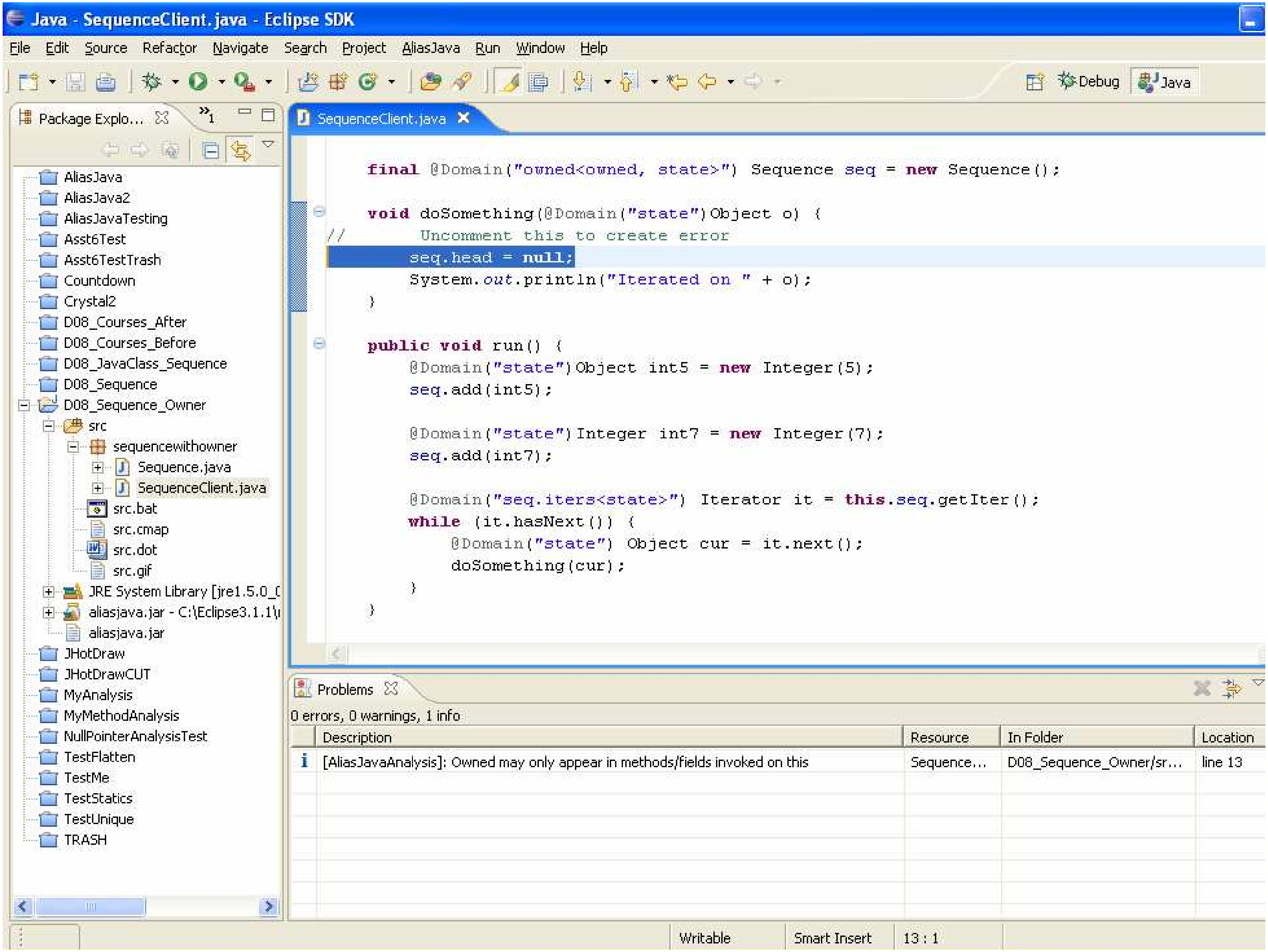


```
@Domain(“seqDom <objDom>”)
Sequence seq = new …
```

- Add domain parameter to hold elements in list
- Link declarations give ***`Sequence.owner`***, ***`Cons.owner`*** ***`(Sequence.owned)`*** access to parameter *`Towner`*

Demo: Sequence and Client Example

- Cannot return list
- Cannot nullify head of list
- Iterate list



```
    final @Domain("owned<owned, state>") Sequence seq = new Sequence();

    void doSomething(@Domain("state")Object o) {
//       Uncomment this to create error
        seq.head = null;
        System.out.println("Iterated on " + o);
    }

    public void run() {
        @Domain("state")Object int5 = new Integer(5);
        seq.add(int5);

        @Domain("state")Integer int7 = new Integer(7);
        seq.add(int7);

        @Domain("seq.iters<state>") Iterator it = this.seq.getIter();
        while (it.hasNext()) {
            @Domain("state") Object cur = it.next();
            doSomething(cur);
        }
    }
```

0 errors, 0 warnings, 1 info

	Description	Resource	In Folder	Location
i	[AliasJavaAnalysis]: Owned may only appear in methods/fields invoked on this	Sequence...	D08_Sequence_Owner/sr...	line 13

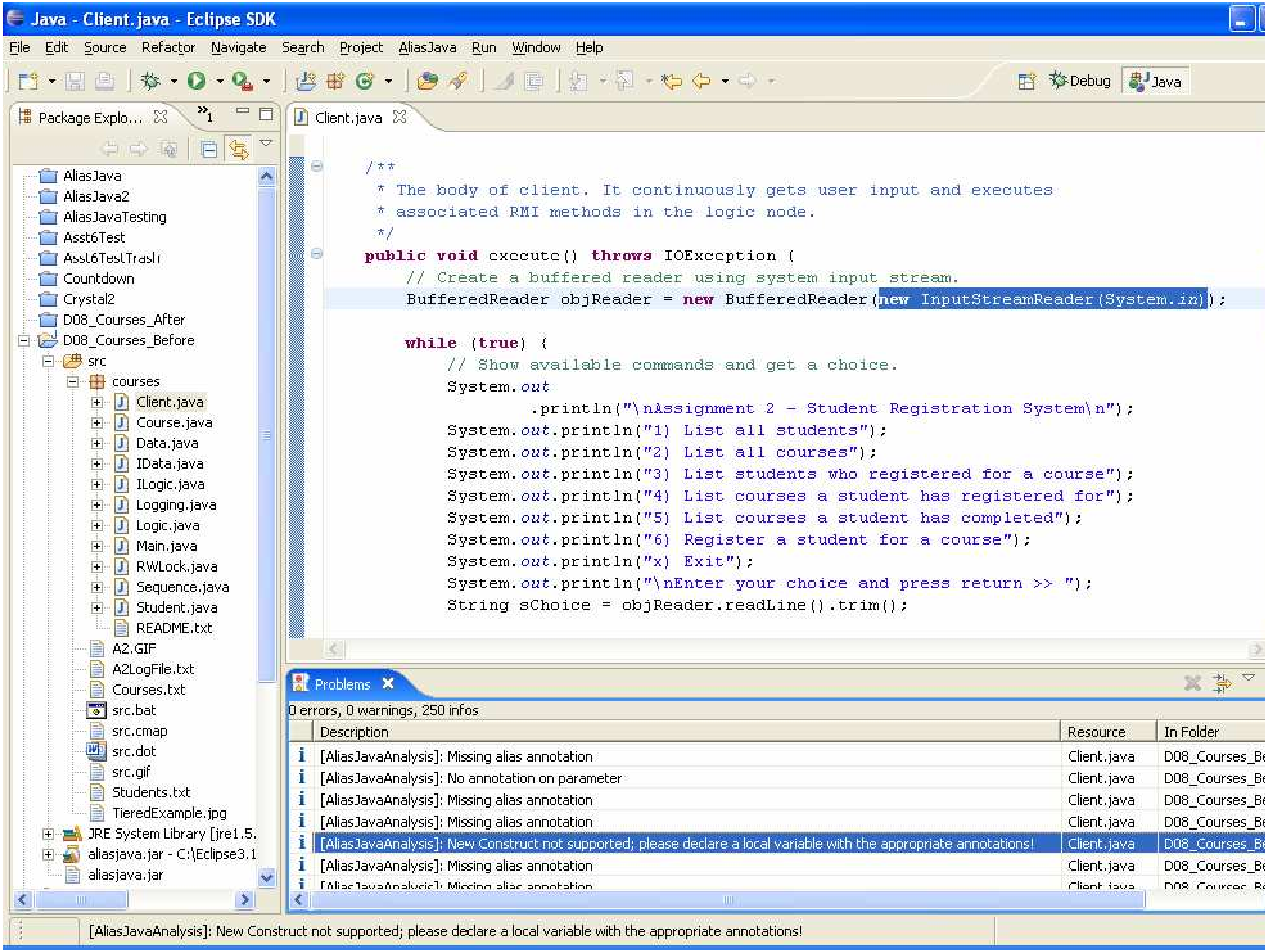
Writable | Smart Insert | 13 : 1

Demo: Course Registration System

- Run analysis on un-annotated program
- Fix problematic coding patterns
- Add default annotations
- Add domain parameters
- Adjust annotations accordingly
- Annotate external code

Problematic coding patterns

- Use string annotation values
- @Target on annotation specifies where annotation allowed (e.g., parameter only)
- Annotations only allowed at declarations
  - Refactor code to declare local variable
  - Add annotation to local variable
  - Some workarounds not very elegant



```
    /**
     * The body of client. It continuously gets user input and executes
     * associated RMI methods in the logic node.
     */
    public void execute() throws IOException {
        // Create a buffered reader using system input stream.
        BufferedReader objReader = new BufferedReader(new InputStreamReader(System.in));

        while (true) {
            // Show available commands and get a choice.
            System.out
                    .println("\nAssignment 2 - Student Registration System\n");
            System.out.println("1) List all students");
            System.out.println("2) List all courses");
            System.out.println("3) List students who registered for a course");
            System.out.println("4) List courses a student has registered for");
            System.out.println("5) List courses a student has completed");
            System.out.println("6) Register a student for a course");
            System.out.println("x) Exit");
            System.out.println("\nEnter your choice and press return >> ");
            String sChoice = objReader.readLine().trim();
```

0 errors, 0 warnings, 250 infos

	Description	Resource	In Folder
i	[AliasJavaAnalysis]: Missing alias annotation	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: No annotation on parameter	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: Missing alias annotation	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: Missing alias annotation	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: New Construct not supported; please declare a local variable with the appropriate annotations!	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: Missing alias annotation	Client.java	D08_Courses_Be
i	[AliasJavaAnalysis]: Missing alias annotation	Client.java	D08_Courses_Be

[AliasJavaAnalysis]: New Construct not supported; please declare a local variable with the appropriate annotations!

Problematic code patterns examples

- Return new Expressions

```
public Iterator getIter() {
    return new SequenceIterator(head);
}
```

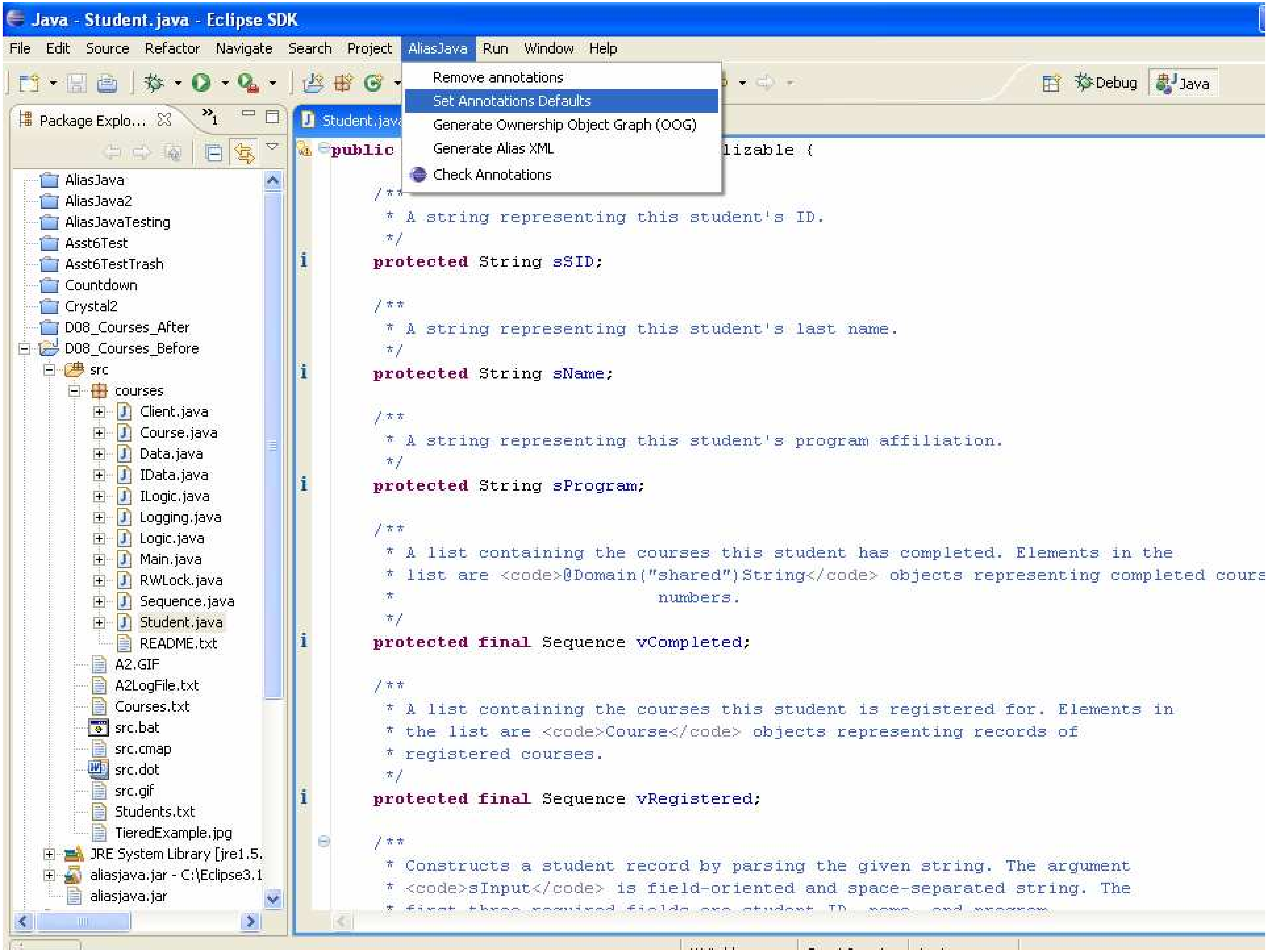
- Cast Expressions

```
ArrayList vCourse = objStudent.getRegisteredCourses();
for (int i=0; i<vCourse.size(); i++) {
    if (((Course) vCourse.get(i)).conflicts(objCourse)) {
            lock.releaseLock();
            return "Registration conflicts";
    }
}
```

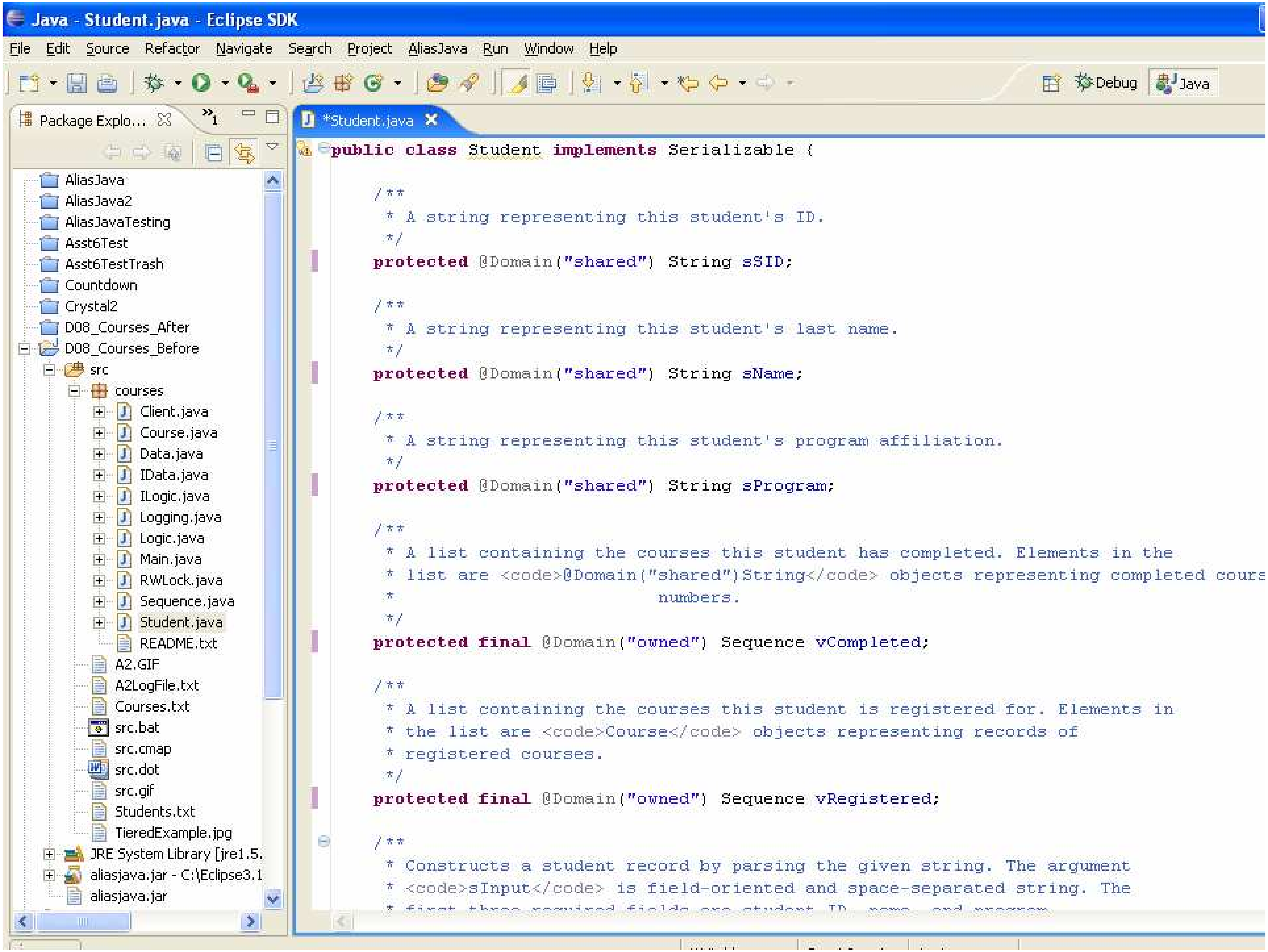
- Anonymous classes, etc.

Adding default annotations

- Reduce annotation burden
  - Strings as “shared”
  - Method parameters as “lent”
  - Private fields as “owned”
- Not a smart “inference” tool
  - Some tools can infer unique and owned
  - Cannot infer domain parameters

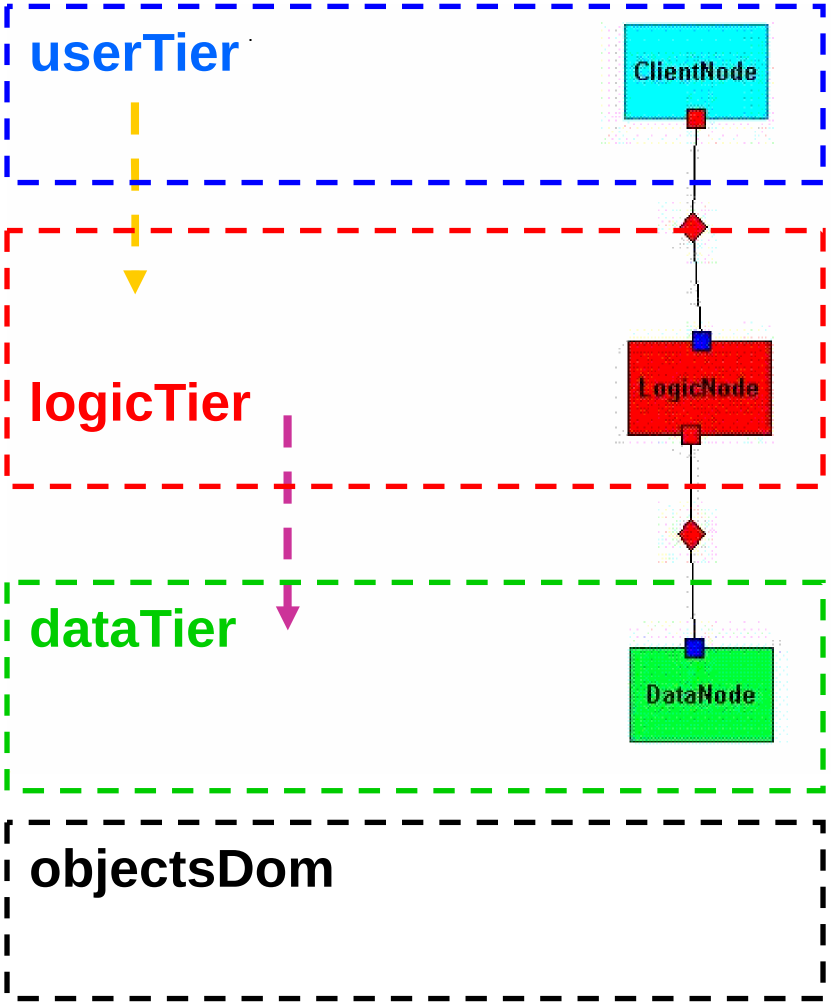


```
public                                          lizable {

    /**
     * A string representing this student's ID.
     */
    protected String sSID;

    /**
     * A string representing this student's last name.
     */
    protected String sName;

    /**
     * A string representing this student's program affiliation.
     */
    protected String sProgram;

    /**
     * A list containing the courses this student has completed. Elements in the
     * list are <code>@Domain("shared")String</code> objects representing completed cours
     *                           numbers.
     */
    protected final Sequence vCompleted;

    /**
     * A list containing the courses this student is registered for. Elements in
     * the list are <code>Course</code> objects representing records of
     * registered courses.
     */
    protected final Sequence vRegistered;

    /**
     * Constructs a student record by parsing the given string. The argument
     * <code>sInput</code> is field-oriented and space-separated string. The
     * first three required fields are student ID, name, and program
```

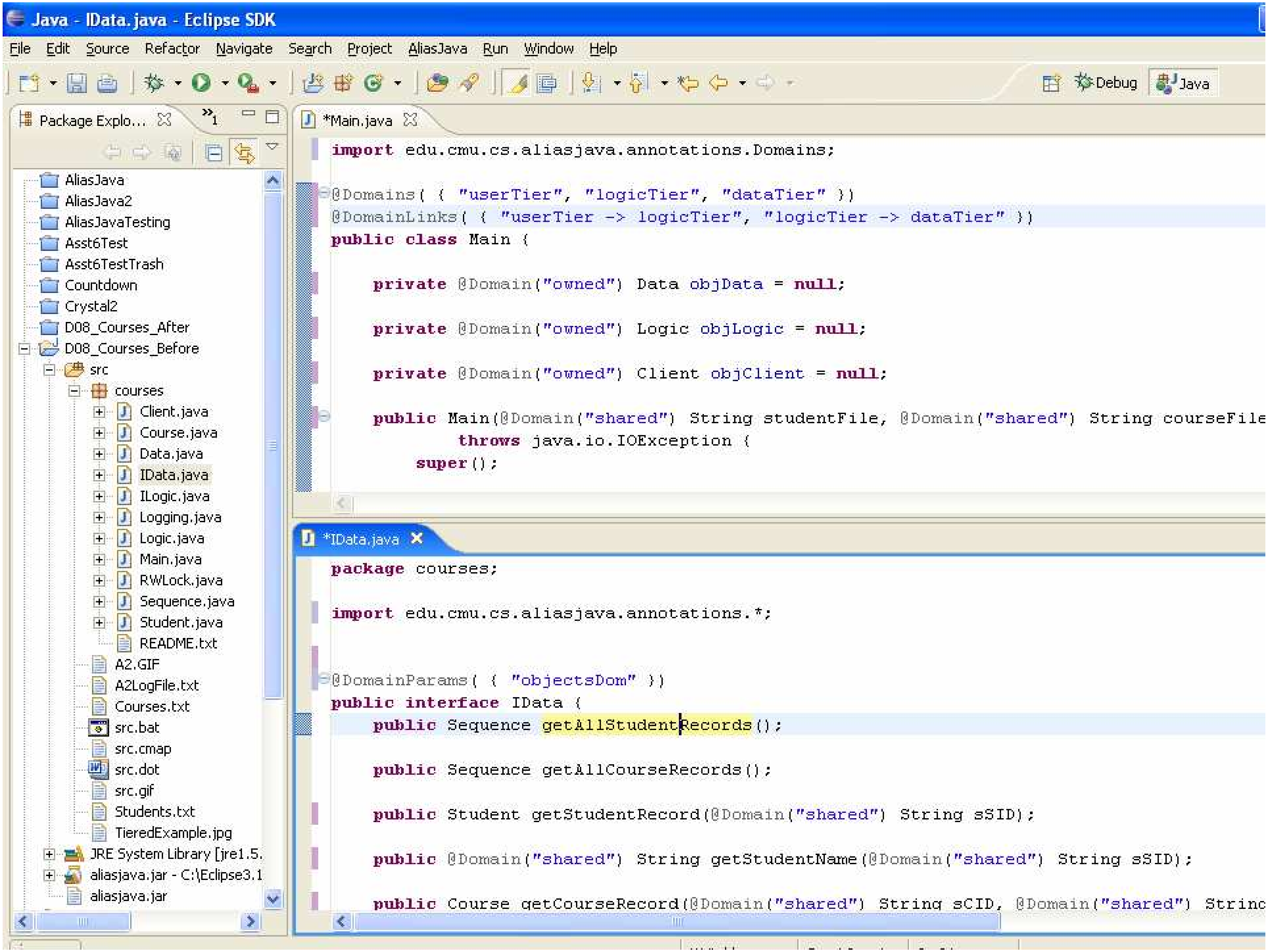


```
public class Student implements Serializable {

    /**
     * A string representing this student's ID.
     */
    protected @Domain("shared") String sSID;

    /**
     * A string representing this student's last name.
     */
    protected @Domain("shared") String sName;

    /**
     * A string representing this student's program affiliation.
     */
    protected @Domain("shared") String sProgram;

    /**
     * A list containing the courses this student has completed. Elements in the
     * list are <code>@Domain("shared")String</code> objects representing completed cours
     *                           numbers.
     */
    protected final @Domain("owned") Sequence vCompleted;

    /**
     * A list containing the courses this student is registered for. Elements in
     * the list are <code>Course</code> objects representing records of
     * registered courses.
     */
    protected final @Domain("owned") Sequence vRegistered;

    /**
     * Constructs a student record by parsing the given string. The argument
     * <code>sInput</code> is field-oriented and space-separated string. The
     * first three required fields are student ID, name, and program
```

Adding domain parameters

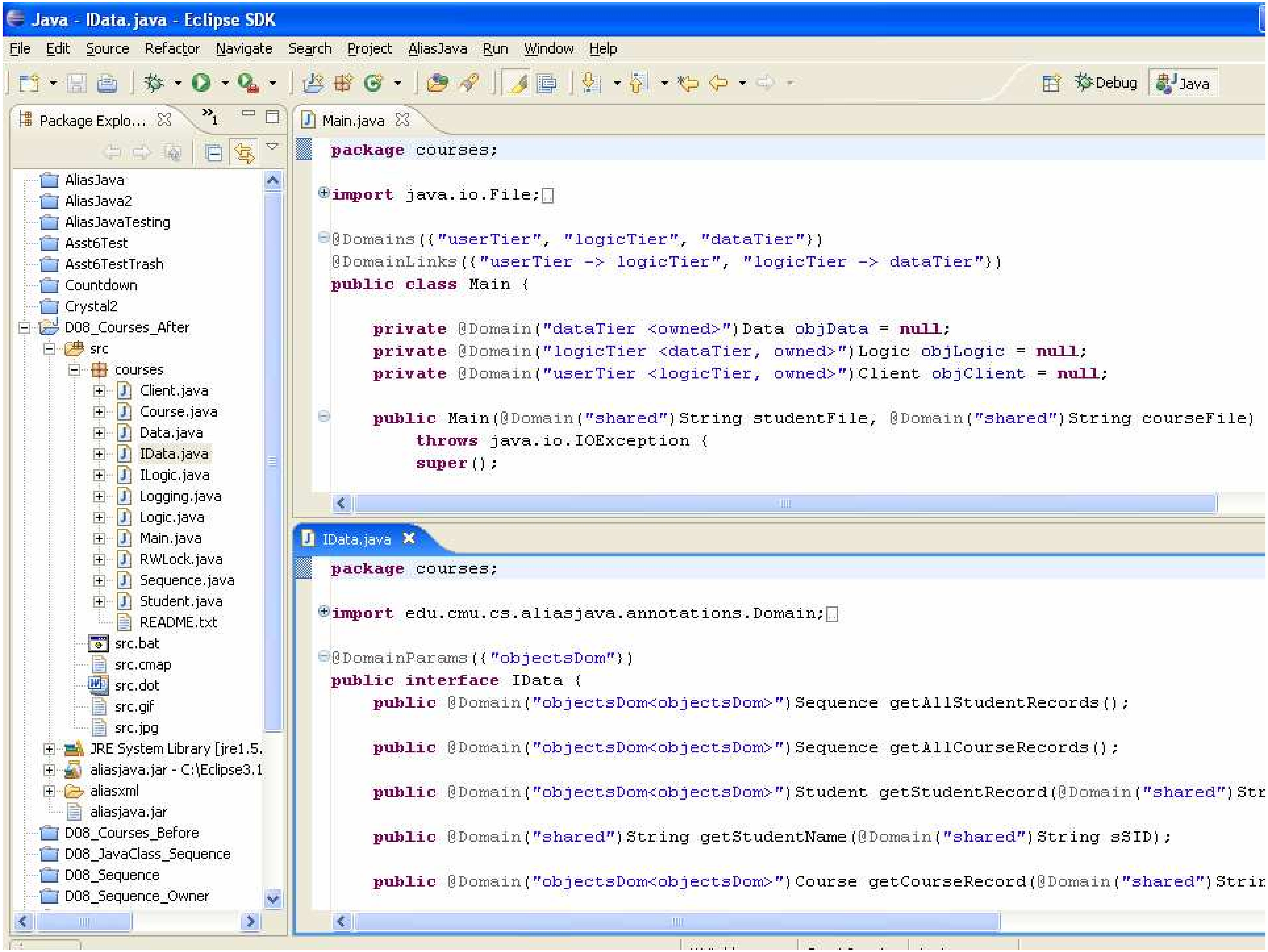


```
@Domains({"userTier",
          "logicTier",
          "dataTier"})
@DomainLinks({"userTier -> logicTier",
              "logicTier -> dataTier"})
public class Main {
  @Domain("dataTier<owned>")
     private Data objData = null;

  @Domain("logicTier<dataTier, owned>")
     private Logic objLogic = null;

  @Domain("userTier<logicTier, owned>")
     private Client objClient = null;
    }
}
```



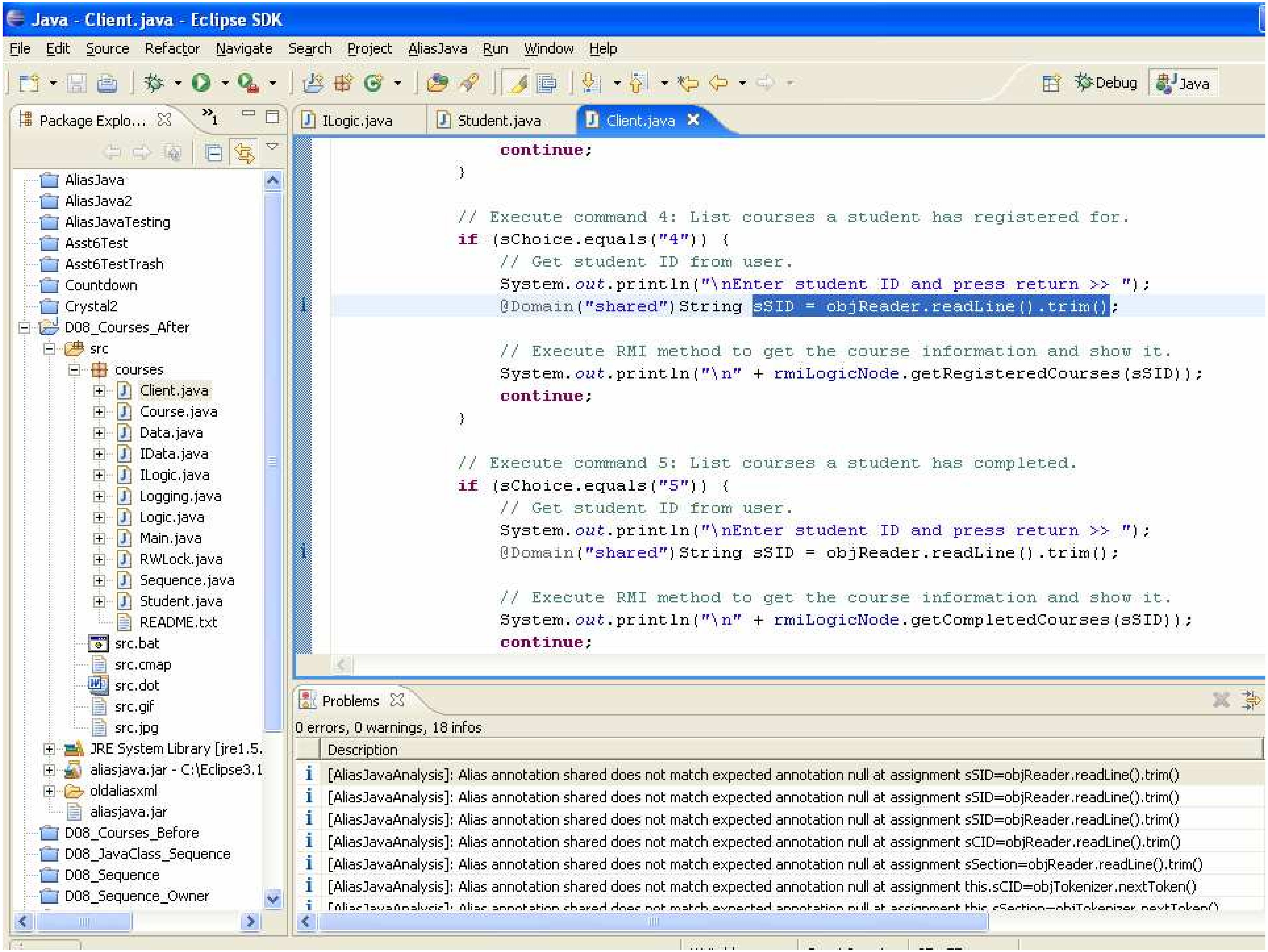
```
import edu.cmu.cs.aliasjava.annotations.Domains;

@Domains( { "userTier", "logicTier", "dataTier" })
@DomainLinks( { "userTier -> logicTier", "logicTier -> dataTier" })
public class Main {

    private @Domain("owned") Data objData = null;

    private @Domain("owned") Logic objLogic = null;

    private @Domain("owned") Client objClient = null;

    public Main(@Domain("shared") String studentFile, @Domain("shared") String courseFile
            throws java.io.IOException {
        super();
```

```
package courses;

import edu.cmu.cs.aliasjava.annotations.*;


@DomainParams( { "objectsDom" })
public interface IData {
    public Sequence getAllStudentRecords();

    public Sequence getAllCourseRecords();

    public Student getStudentRecord(@Domain("shared") String sSID);

    public @Domain("shared") String getStudentName(@Domain("shared") String sSID);

    public Course getCourseRecord(@Domain("shared") String sCID, @Domain("shared") String
```



```
package courses;

import java.io.File;

@Domains({"userTier", "logicTier", "dataTier"})
@DomainLinks({"userTier -> logicTier", "logicTier -> dataTier"})
public class Main {

    private @Domain("dataTier <owned>")Data objData = null;
    private @Domain("logicTier <dataTier, owned>")Logic objLogic = null;
    private @Domain("userTier <logicTier, owned>")Client objClient = null;

    public Main(@Domain("shared")String studentFile, @Domain("shared")String courseFile)
        throws java.io.IOException {
        super();
```

```
package courses;

import edu.cmu.cs.aliasjava.annotations.Domain;

@DomainParams({"objectsDom"})
public interface IData {
    public @Domain("objectsDom<objectsDom>")Sequence getAllStudentRecords();

    public @Domain("objectsDom<objectsDom>")Sequence getAllCourseRecords();

    public @Domain("objectsDom<objectsDom>")Student getStudentRecord(@Domain("shared")Str

    public @Domain("shared")String getStudentName(@Domain("shared")String sSID);

    public @Domain("objectsDom<objectsDom>")Course getCourseRecord(@Domain("shared")Strin
```



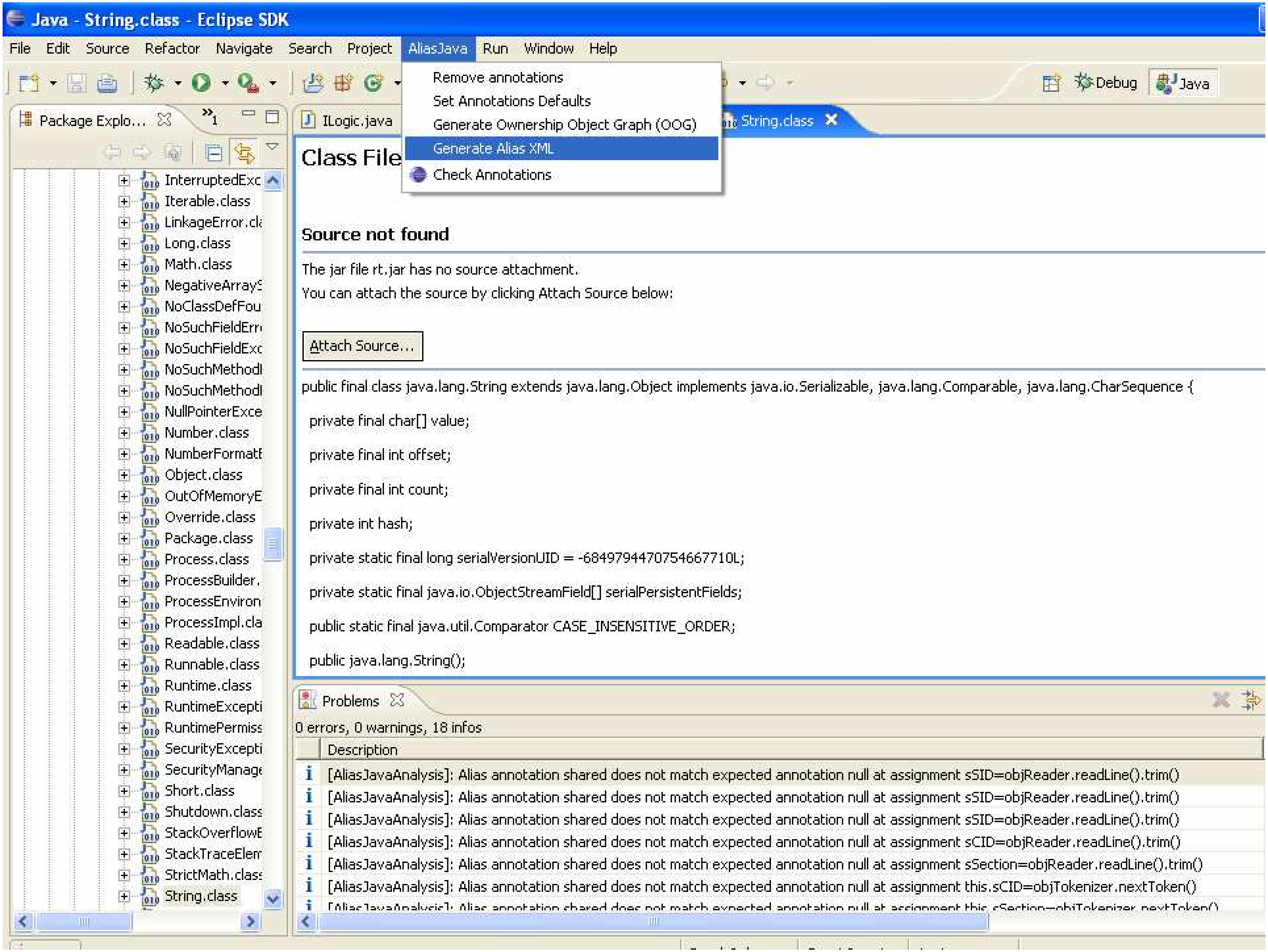
```java
        continue;
    }

    // Execute command 4: List courses a student has registered for.
    if (sChoice.equals("4")) {
        // Get student ID from user.
        System.out.println("\nEnter student ID and press return >> ");
        @Domain("shared")String sSID = objReader.readLine().trim();

        // Execute RMI method to get the course information and show it.
        System.out.println("\n" + rmiLogicNode.getRegisteredCourses(sSID));
        continue;
    }

    // Execute command 5: List courses a student has completed.
    if (sChoice.equals("5")) {
        // Get student ID from user.
        System.out.println("\nEnter student ID and press return >> ");
        @Domain("shared")String sSID = objReader.readLine().trim();

        // Execute RMI method to get the course information and show it.
        System.out.println("\n" + rmiLogicNode.getCompletedCourses(sSID));
        continue;
```

0 errors, 0 warnings, 18 infos

```
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sCID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSection=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sCID=objTokenizer.nextToken()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sSection=objTokenizer.nextToken()
```

Annotating external code

- Need to annotate code outside of source
    - Standard JDK library
    - Other third-party libraries
- Ideally, library provider adds annotations
- Annotated only parts of library in use
- Annotations shared amongst authors
- Wizard to generate skeleton XML file
    - Place annotations in XML file (AliasXML)
    - No semantic differences

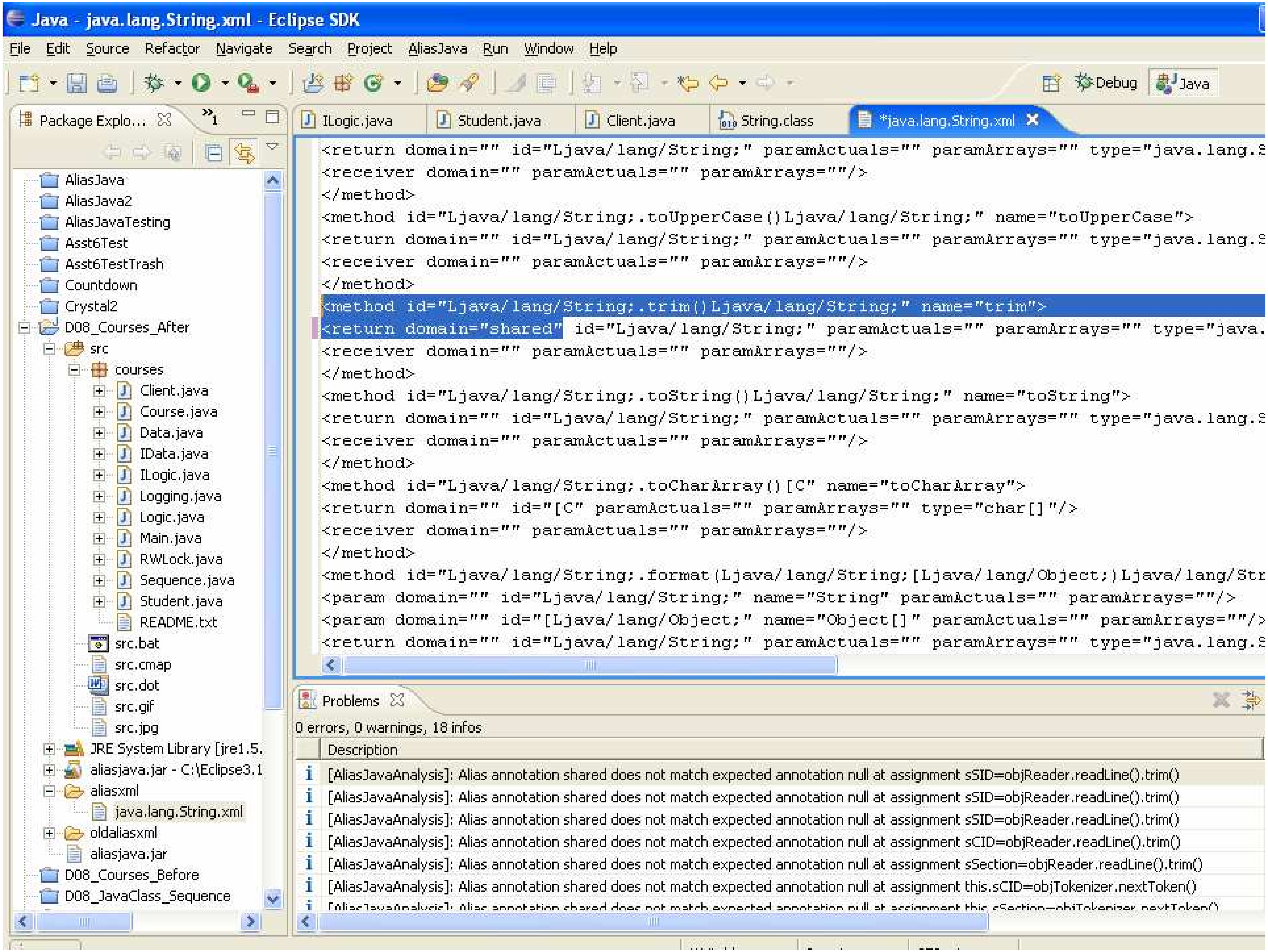


Source not found

The jar file rt.jar has no source attachment.
You can attach the source by clicking Attach Source below:

```
public final class java.lang.String extends java.lang.Object implements java.io.Serializable, java.lang.Comparable, java.lang.CharSequence {

 private final char[] value;

 private final int offset;

 private final int count;

 private int hash;

 private static final long serialVersionUID = -6849794470754667710L;

 private static final java.io.ObjectStreamField[] serialPersistentFields;

 public static final java.util.Comparator CASE_INSENSITIVE_ORDER;

 public java.lang.String();
```

```
0 errors, 0 warnings, 18 infos
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sCID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSection=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sCID=objTokenizer.nextToken()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sSection=objTokenizer.nextToken()
```



```
<return domain="" id="Ljava/lang/String;" paramActuals="" paramArrays="" type="java.lang.S
<receiver domain="" paramActuals="" paramArrays=""/>
</method>
<method id="Ljava/lang/String;.toUpperCase()Ljava/lang/String;" name="toUpperCase">
<return domain="" id="Ljava/lang/String;" paramActuals="" paramArrays="" type="java.lang.S
<receiver domain="" paramActuals="" paramArrays=""/>
</method>
<method id="Ljava/lang/String;.trim()Ljava/lang/String;" name="trim">
<return domain="shared" id="Ljava/lang/String;" paramActuals="" paramArrays="" type="java.
<receiver domain="" paramActuals="" paramArrays=""/>
</method>
<method id="Ljava/lang/String;.toString()Ljava/lang/String;" name="toString">
<return domain="" id="Ljava/lang/String;" paramActuals="" paramArrays="" type="java.lang.S
<receiver domain="" paramActuals="" paramArrays=""/>
</method>
<method id="Ljava/lang/String;.toCharArray()[C" name="toCharArray">
<return domain="" id="[C" paramActuals="" paramArrays="" type="char[]"/>
<receiver domain="" paramActuals="" paramArrays=""/>
</method>
<method id="Ljava/lang/String;.format(Ljava/lang/String;[Ljava/lang/Object;)Ljava/lang/Str
<param domain="" id="Ljava/lang/String;" name="String" paramActuals="" paramArrays=""/>
<param domain="" id="[Ljava/lang/Object;" name="Object[]" paramActuals="" paramArrays=""/>
<return domain="" id="Ljava/lang/String;" paramActuals="" paramArrays="" type="java.lang.S
```

0 errors, 0 warnings, 18 infos

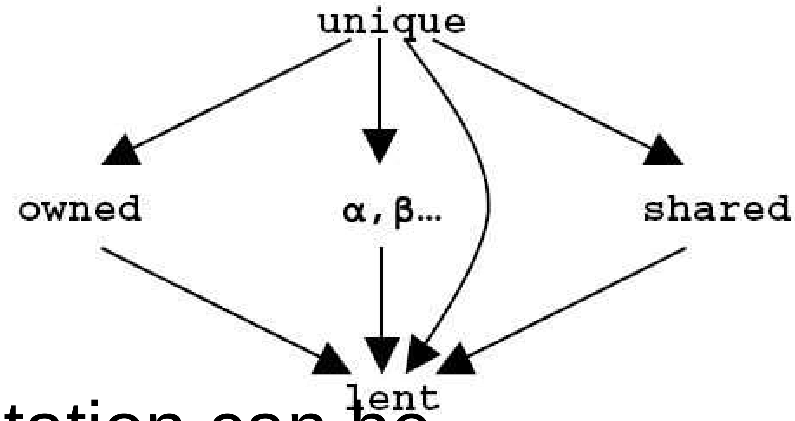
```
Description
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sCID=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment sSection=objReader.readLine().trim()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sCID=objTokenizer.nextToken()
i [AliasJavaAnalysis]: Alias annotation shared does not match expected annotation null at assignment this.sSection=objTokenizer.nextToken()
```

Sample rules (method declaration)

- Check return type annotation
  - If reference type, must have an annotation
  - May not be marked **owned** for public methods
- Check parameter annotations
  - If reference type, must have an annotation
  - May not be marked **owned** for public methods
- Check overriding
  - May not change return type annotation
  - May not change parameter annotation
  - May not change receiver annotation

Checking assignment rule

- Arrow means data can flow between variables with two annotations



- Variable with any type annotation can be assigned a **unique** value
- **lent** variables can be assigned a value with any type annotation
- Values with type annotations **owned** and **shared**, as well as declared domains must be kept separate from each other

Future work

- Ease restrictions on coding constructs
  - Inter-procedural annotation inference
  - E.g., allow { return new …() }
- Integrate other kinds of annotations
  - @Domain(“extunique”): externally unique
  - @Domain(“readonly”): immutable
- Integrate interactive annotation inference
  - Determining ownership parameters difficult
  - Using annotations does not break the code

Implementation Status

- Eclipse Plug-in

- For more information
  - Related Demonstration “**A Static Analysis for Extracting Runtime Views from Annotated Object-Oriented Code**”
  - http://www.archjava.org

Summary

- Re-implemented ownership domains as annotations using Java 1.5
- Used Eclipse JDT
- Using annotations to improve adoptability
  - Better tool support
  - Incrementally and partially specifying annotations on large code bases
  - More annotations in a non-breaking way