

Introduction to Programming I

Lab 14

Alexey Shikulin
Furqan Haider
Munir Makhmutov
Sami Sellami

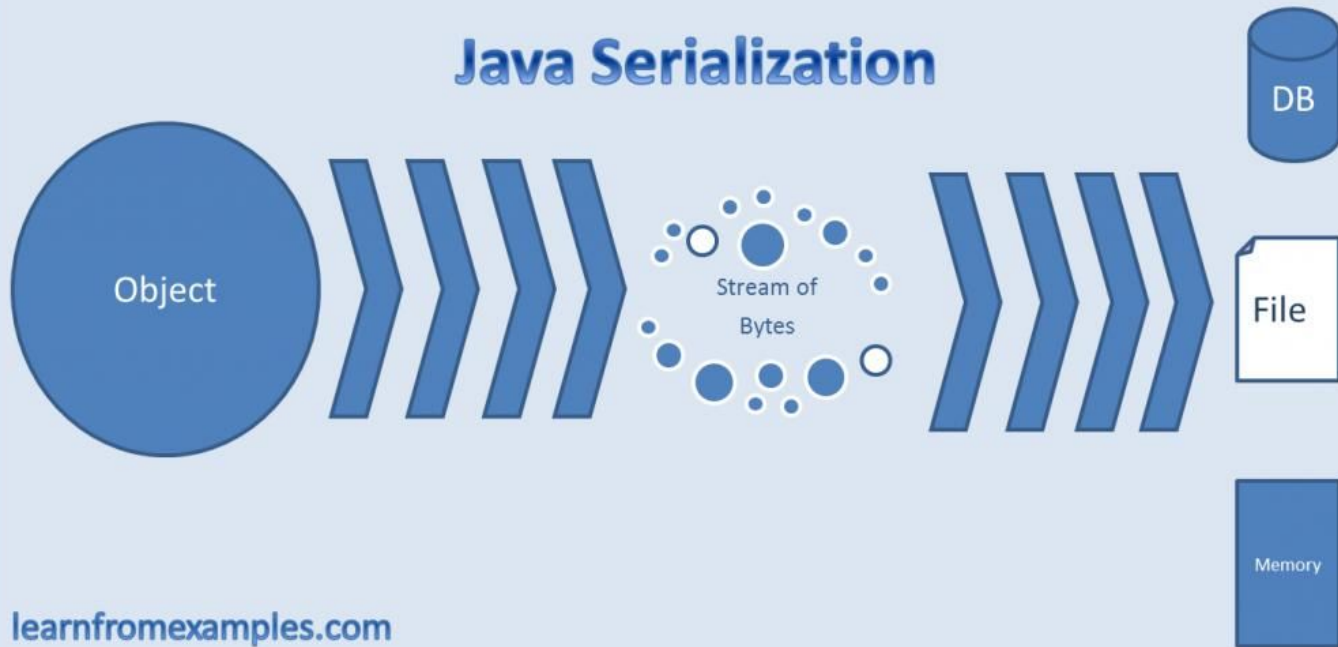
Agenda

- Recap of the topic
- Warm-up exercises
- Solving Problems
- Q&A

Learning outcome:

- Enhance your knowledge on:
 - Serialization
 - Reflection
- Prepare for interview questions
- Practical / applicable knowledge

Java Serialization



Serialization

To serialize an object means to convert its state to a byte stream so that the byte stream can be reverted back into a copy of the object.

A Java object is serializable if its class or any of its superclasses implements either the **java.io.Serializable** interface or its subinterface, **java.io.Externalizable**.

Deserialization is the process of converting the serialized form of an object back into a copy of the object.



Serialization

```
public class Employee implements java.io.Serializable {  
    public String name;  
    public String address;  
    public transient int SSN;  
    public int number;  
  
    public void mailCheck() {  
        System.out.println("Mailing a check to " + name + " " + address);  
    }  
}
```

Serialization

```
import java.io.*;
public class SerializeDemo {

    public static void main(String [] args) {
        Employee e = new Employee();
        e.name = "Reyan Ali";
        e.address = "Phokka Kuan, Ambehta Peer";
        e.SSN = 11122333;
        e.number = 101;

        try {
            FileOutputStream fileOut =
                new FileOutputStream("/tmp/employee.ser");
            ObjectOutputStream out = new ObjectOutputStream(fileOut);
            out.writeObject(e);
            out.close();
            fileOut.close();
            System.out.printf("Serialized data is saved in /tmp/employee.ser");
        } catch (IOException i) {
            i.printStackTrace();
        }
    }
}
```

Deserialization

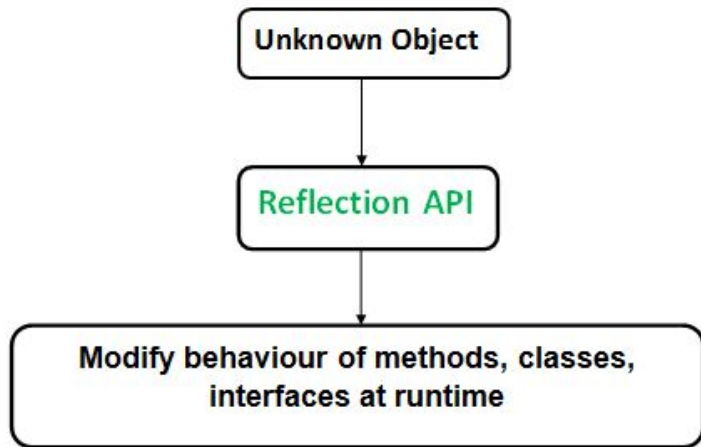
After a serialized object has been written into a file, it can be read from the file and deserialized, that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.

Deserialization

```
import java.io.*;
public class DeserializeDemo {
    public static void main(String [] args) {
        Employee e = null;
        try {
            FileInputStream fileIn = new FileInputStream("/tmp/employee.ser");
            ObjectInputStream in = new ObjectInputStream(fileIn);
            e = (Employee) in.readObject();
            in.close();
            fileIn.close();
        } catch (IOException i) {
            i.printStackTrace();
            return;
        } catch (ClassNotFoundException c) {
            System.out.println("Employee class not found");
            c.printStackTrace();
            return;
        }
        System.out.println("Deserialized Employee...");
        System.out.println("Name: " + e.name);
        System.out.println("Address: " + e.address);
        System.out.println("SSN: " + e.SSN);
        System.out.println("Number: " + e.number);
    }
}
```


Java Reflection

Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its members and display them.



Reflection

```
// class whose object is to be created
class Test
{
    // creating a private field
    private String s;

    // creating a public constructor
    public Test() { s = "GeeksforGeeks"; }

    // Creating a public method with no arguments
    public void method1() {
        System.out.println("The string is " + s);
    }

    // Creating a public method with int as argument
    public void method2(int n) {
        System.out.println("The number is " + n);
    }

    // creating a private method
    private void method3() {
        System.out.println("Private method invoked");
    }
}
```

Reflection Example

```
class Demo {
    public static void main(String[] args) throws Exception {
        Test obj = new Test();
        // Creating class object from the object using getClass method
        Class<?> cls = obj.getClass();
        System.out.println("The name of class is " + cls.getName());
        // Getting the constructor of the class through the object of the class
        Constructor<?> constructor = cls.getConstructor();
        System.out.println("The name of constructor is " + constructor.getName());
        System.out.println("The public methods of class are: ");
        // Getting methods of the class through the object of the class by using getMethods
        Method[] methods = cls.getMethods();
        // Printing method names
        for (Method method : methods)
            System.out.println(method.getName());
        // creates object of desired method by providing the method name and parameter class as
        // arguments to the getDeclaredMethod
        Method methodCall1 = cls.getDeclaredMethod("method2", int.class);
        // invokes the method at runtime
        methodCall1.invoke(obj, 19);
    }
}
```

Reflection Example

```
class Demo {
    public static void main(String[] args) throws Exception {
        Test obj = new Test();
        // Creating class object from the object using getClass method
        Class<?> cls = obj.getClass();
        System.out.println("The name of class is " + cls.getName());
        // Getting the constructor of the class through the object of the class
        Constructor<?> constructor = cls.getConstructor();
        System.out.println("The name of constructor is " + constructor.getName());
        System.out.println("The public methods of class are: ");
        // Getting methods of the class through the object of the class by using getMethods
        Method[] methods = cls.getMethods();
        // Printing method names
        for (Method method : methods)
            System.out.println(method.getName());
        // creates object of desired method by providing the method name and parameter class as
        // arguments to the getDeclaredMethod
        Method methodCall1 = cls.getDeclaredMethod("method2", int.class);
        // invokes the method at runtime
        methodCall1.invoke(obj, 19);
    }
}
```

Reflection Example

```
class Demo {
    public static void main(String[] args) throws Exception {
        Test obj = new Test();
        // Creating class object from the object using getClass method
        Class<?> cls = obj.getClass();
        System.out.println("The name of class is " + cls.getName());
        // Getting the constructor of the class through the object of the class
        Constructor<?> constructor = cls.getConstructor();
        System.out.println("The name of constructor is " + constructor.getName());
        System.out.println("The public methods of class are: ");
        // Getting methods of the class through the object of the class by using getMethods
        Method[] methods = cls.getMethods();
        // Printing method names
        for (Method method : methods)
            System.out.println(method.getName());
        // creates object of desired method by providing the method name and parameter class as x
        // arguments to the getDeclaredMethod
        Method methodCall1 = cls.getDeclaredMethod("method2", int.class);
        // invokes the method at runtime
        methodCall1.invoke(obj, 19);
    }
}
```

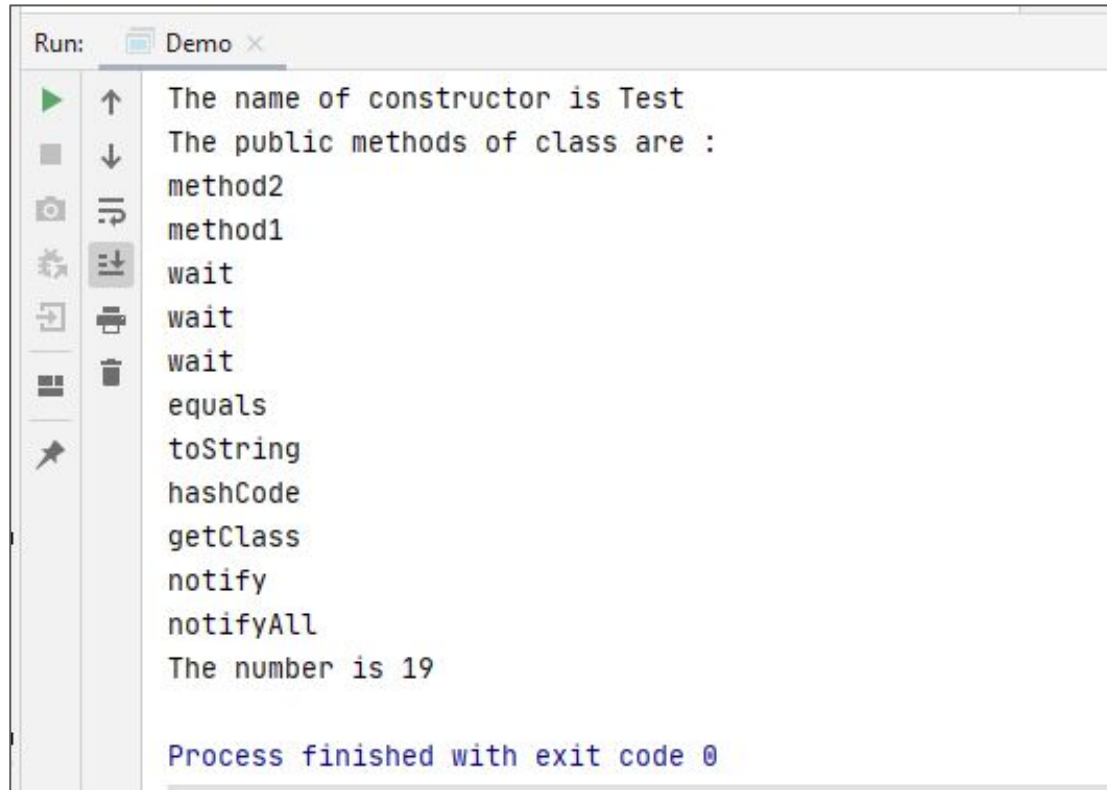
Reflection Example

```
class Demo {
    public static void main(String[] args) throws Exception {
        Test obj = new Test();
        // Creating class object from the object using getClass method
        Class<?> cls = obj.getClass();
        System.out.println("The name of class is " + cls.getName());
        // Getting the constructor of the class through the object of the class
        Constructor<?> constructor = cls.getConstructor();
        System.out.println("The name of constructor is " + constructor.getName());
        System.out.println("The public methods of class are: ");
        // Getting methods of the class through the object of the class by using getMethods
        Method[] methods = cls.getMethods();
        // Printing method names
        for (Method method : methods)
            System.out.println(method.getName());
        // creates object of desired method by providing the method name and parameter class as
        // arguments to the getDeclaredMethod
        Method methodCall1 = cls.getDeclaredMethod("method2", int.class);
        // invokes the method at runtime
        methodCall1.invoke(obj, 19);
    }
}
```

Reflection Example

```
class Demo {  
    public static void main(String[] args) throws Exception {  
        Test obj = new Test();  
        // Creating class object from the object using getClass method  
        Class<?> cls = obj.getClass();  
        System.out.println("The name of class is " + cls.getName());  
        // Getting the constructor of the class through the object of the class  
        Constructor<?> constructor = cls.getConstructor();  
        System.out.println("The name of constructor is " + constructor.getName());  
        System.out.println("The public methods of class are: ");  
        // Getting methods of the class through the object of the class by using getMethods  
        Method[] methods = cls.getMethods();  
        // Printing method names  
        for (Method method : methods)  
            System.out.println(method.getName());  
        // creates object of desired method by providing the method name and parameter class as  
        // arguments to the getDeclaredMethod  
        Method methodCall1 = cls.getDeclaredMethod("method2", int.class);  
        // invokes the method at runtime  
        methodCall1.invoke(obj, 19);  
    }  
}
```

Reflection



The screenshot shows a 'Run' window with a tab labeled 'Demo'. On the left is a vertical toolbar with icons for running, stepping through code, and other debugging actions. The main area contains the following text output:

```
The name of constructor is Test
The public methods of class are :
method2
method1
wait
wait
wait
equals
toString
hashCode
getClass
notify
notifyAll
The number is 19

Process finished with exit code 0
```


Warm-Up Exercises

What conditions must be met in order to serialize a class successfully?

Warm-Up Exercises

The transient keyword in Java is used to indicate that a field should not be part of the serialization (which means saved, like to a file) process.

When to use transient fields?

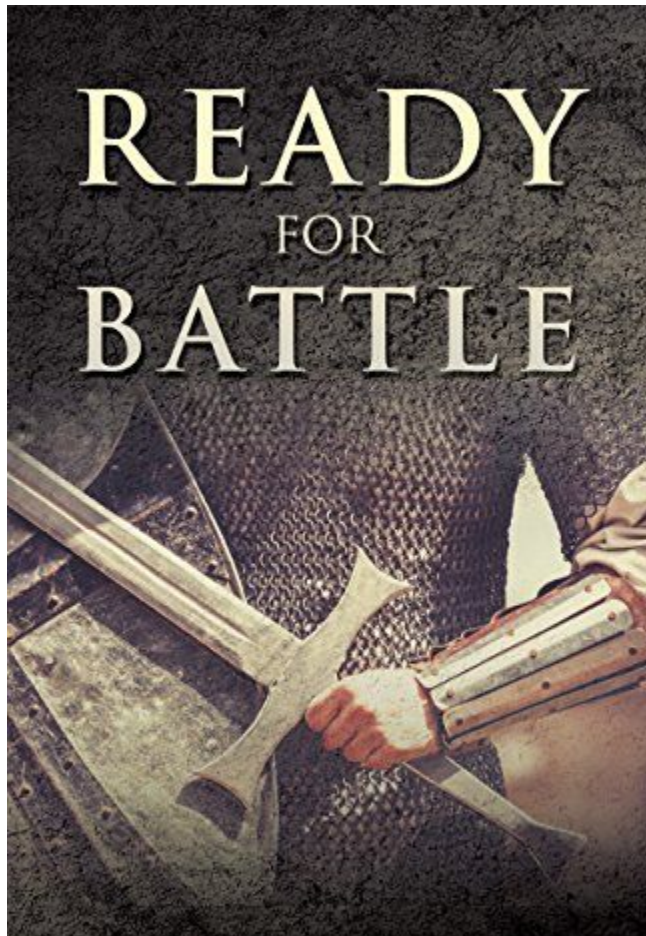
Warm-Up Exercises

Should the *private static final long serialVersionUID* field be initialized for Serializable classes?

Warm-Up Exercises

What are the disadvantages of Reflection?

READY FOR BATTLE



Exercise 1: Serialization (1/2)

- You have to implement a part of online book reading service, which remembers the books read by a single user (do NOT implement class for the user). For this you require 2 classes: *Book* and *ReaderLibrary*
- Create *Book* class with *author*, *title*, *issueYear*, *pageNumber* and *bookmark* and implement *Serializable* interface
- Create *ReaderLibrary* class that will have next methods:
 - *addMyReadBooks(List<Book> books, String serializedFilePath)*
// it should allow to add read books to serialization file (*serialized.dat*)
 - *getMySerializedBooks()* // it should get all read books from serialization file (*serialized.dat*)
 - *displayMyBooks(List<Book> books)* // it should display all books info
- Do NOT forget to add next string to *ReaderLibrary* class:
 - *private static final long serialVersionUID = 1L;* // this is unique number for serialization file
// do not modify name and value

Exercise 1: Serialization (2/2)

- To test your program run 2 methods:
 - *addMyReadBooks(List<Book> books, int booksNumber, String serializedFilePath)*
 - *displayMyBooks(List<Book> books)*
- Quit and run another sequence of methods:
 - *getMySerializedBooks()*
 - *displayMyBooks(List<Book> books)*
- The outputs have to be the same. It will demonstrate that you were able to save the state of reader's progress of reading books

Exercise 2: Reflection

- Your task is to implement a program that examines class Code (see the next slide), and print out the interface for the given class. The Reflection API must be used. The interface must include:
 - Method headers
 - Return value data types
 - Parameter data types

Exercise 1: Reflection (cont.)

```
class Code
{
    // a private field
    private String value;

    // a public constructor
    public Code() { value = "Nothing is there yet"; }

    // getter for value
    public String getValue() {
        return value;
    }

    // setter for value
    public void setValue(String newValue) {
        value = newValue;
    }

    // private method
    private void printToTerminal() {
        System.out.println(value);
    }
}
```

References

- https://www.tutorialspoint.com/java/java_serialization.htm
- <https://www.oracle.com/technical-resources/articles/java/javareflection.html>