Java IO. Part II. Object serialization

Lecture 22

Summary: streams

- Core connection streams represent a connection to a source or a sink of data. They are low-level classes which do an actual work
- Chain streams (decorators) wrap around connection streams, cannot work on their own, need to be chained to connection streams

Main utilities of java.io

- File manipulation
- Writing and reading of Streams
- Serializing objects

Saving object state

- Objects have state and behavior
- Behavior is defined in the class, but state lives with each individual object

- How to save and restore the object state?

Two ways of saving the state

- Write the value of each variable into a file (if the data to be used by a non-java application)
- Serialize the entire object (if the state to be used only inside java code)

1. Writing to a stream as a plain text file with delimiters

Create a file and write one line per character, each character property separated by a delimiter:

```
BufferedWriter writer =
        new BufferedWriter(new FileWriter(“game.txt”));
        writer.write(unit.getPower() +
                              “,”+unit.getName()+…“\n”);
```

This results in the following output file:

50,Elf,bow, sword,dust
200,Troll,bare hands,big ax
120,Magician,spells,invisibility

Which is easy for human to read

Reading from a stream

```
File myFile = new File(“game.txt”);
FileReader fileReader = new
            FileReader(myFile);
BufferedReader reader =
      new BufferedReader(fileReader);
String line = null;
while ((line = reader.readLine()) != null)
{
      System.out.println(line);
}
reader.close();
```

Object serialization

Object on the heap

The object has state:
values of instance variables

Object serialized

Saves the values of instance variables,
so the identical object can be restored

2. Serialize entire character object

```
GameCharacter one = new GameCharacter(50, “Elf”, new
                String[] {“bow”, “sword”, “dust”});
GameCharacter two = new GameCharacter(200, “Troll”, new
                String[] {“bare hands”, “big ax”});
GameCharacter three = new GameCharacter(120, “Magician”,
                new String[] {“spells”, “invisibility”});

// imagine code that does things with the characters that might change their
state values
ObjectOutputStream os = new ObjectOutputStream(
                        new FileOutputStream(“Game.ser”));
os.writeObject(one);
os.writeObject(two);
os.writeObject(three);
os.close();
```

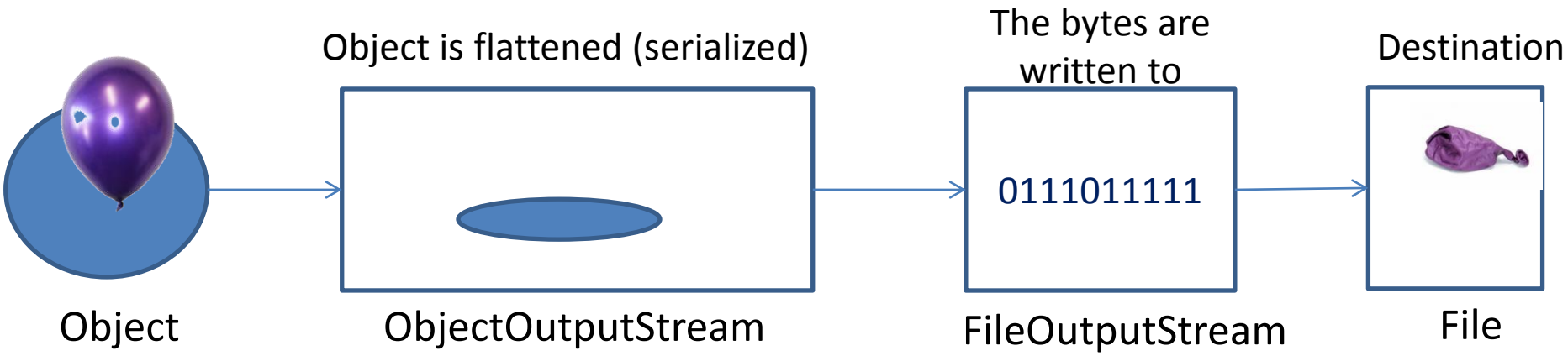
Much easier for Java program to read

De-serialize

```
ObjectInputStream is = new ObjectInputStream(
                       new FileInputStream("Game.ser"));
GameCharacter oneRestore
              = (GameCharacter) is.readObject();
GameCharacter twoRestore
              = (GameCharacter) is.readObject();
GameCharacter threeRestore
              = (GameCharacter) is.readObject();
```

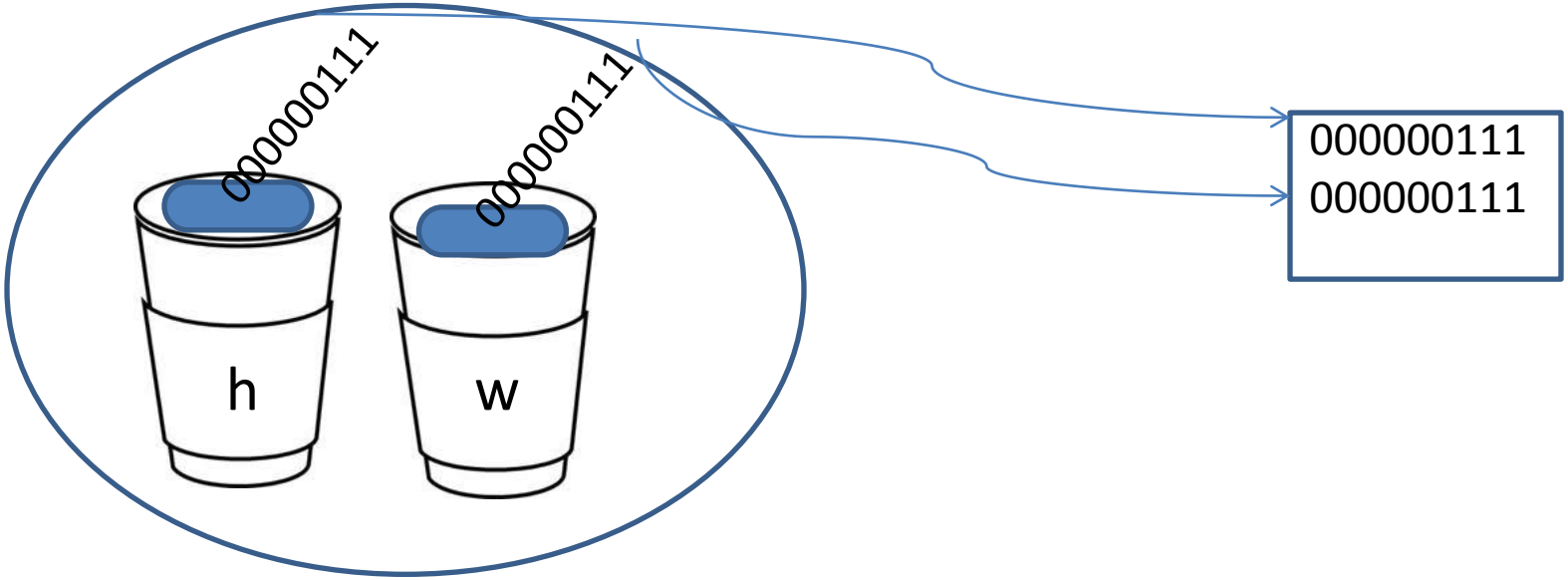
ObjectOutputStream

- Converts **objects** to a sequence of bytes, which are passed to the underlying connection stream (for example FileOutputStream)
- FileOutputStream writes bytes to a file



What happens to an object when it is serialized

The values of all instance variables are converted into a sequence of bytes



What needs to be saved?

- Primitive variables contain values
- Reference variables contain object references
- The objects these variables refer to may have their own instance variables that are reference variables

The entire object graph is serialized

- When the main object is serialized, all the objects it refers to are also serialized, and all the objects these objects refer to…
- Serialization saves the entire object graph
- This happens automatically
- If a child object in the object graph is referenced by more than one reference variable, it is saved only once.

Implement ***Serializable***

- Serializable – marker (**tag**) interface
- Objects of Serializable are saveable through serialization mechanism

All or nothing gets serialized

- Either the entire object graph is serialized correctly, or serialization fails (*NotSerializable* Exception)
- You can’t serialize a Pond object if its Duck variable refuses to be serialized by not implementing *Serializable*

Transient keyword

- Mark an instance variable as ***transient*** if it can't or should not be serialized.
- The run-time specific information should not be saved (network connections, threads, file objects…). Once the program shuts down, there is no way to bring these things back to life in any meaningful way.
- These should be recreated from scratch each time

Transient fields example

```
class Chat implements Serializable {
        transient String currentID;
        String userName;
        // more code
}
```

When the object is brought back, the transient fields are initialized to default values (null for instance variables)

Serializable or not?

- All subclasses of a serializable class are serializable by default (normal inheritance)

- If class designer forgot to make its class serializable, you can subclass and make your subclass serializable.
- In this case, when the subclassed object gets deserialized, the constructor of its superclass (which is not Serializable) runs as if for new object creation of a superclass

Why not serializable by default

- Can’t un-serialize
- Some objects should not be saved: password
- Some things does not make sense to save

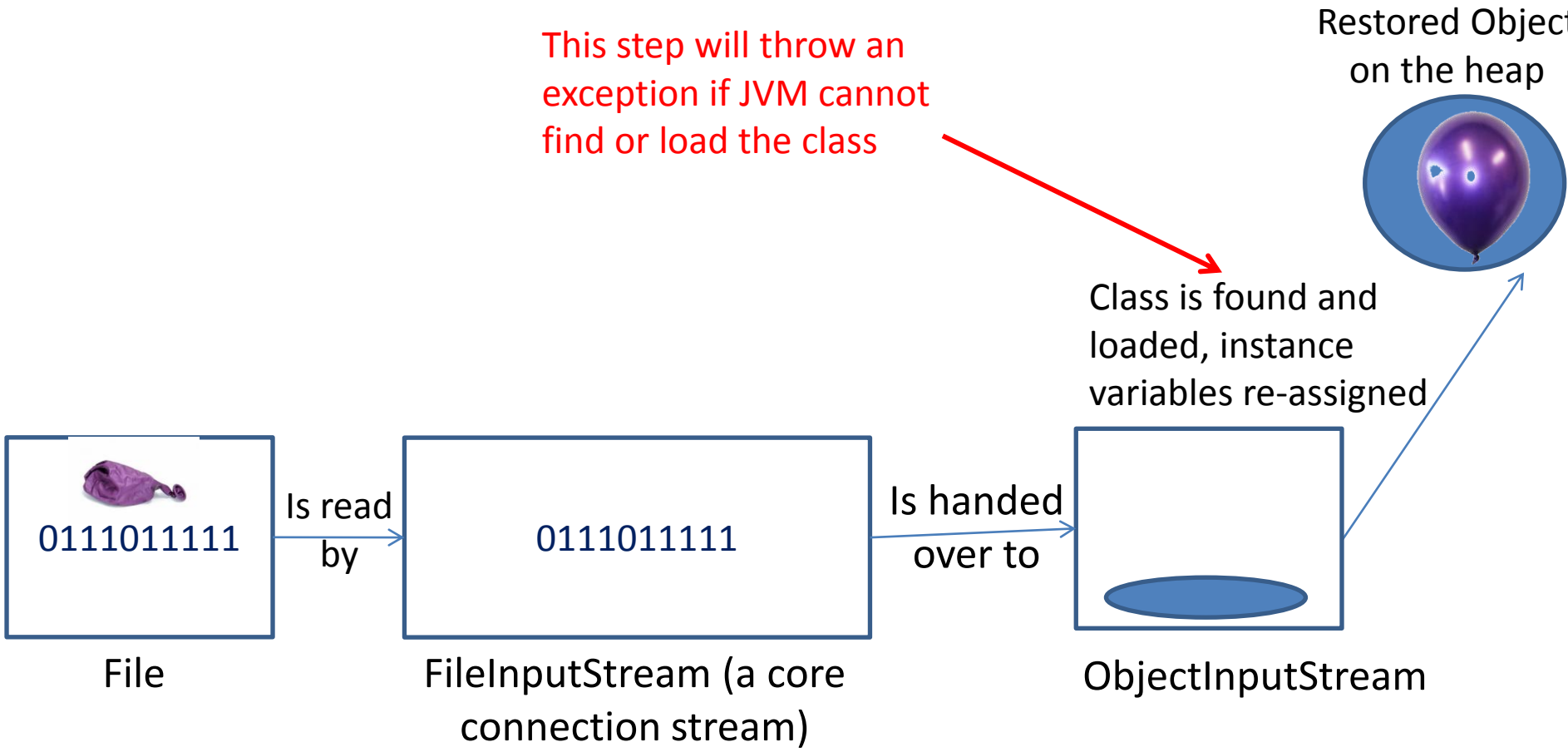
Deserialization: restoring the object

```
FileInputStream fileStream = new
     FileInputStream(“MyGame.ser”);
ObjectInputStream os = new
     ObjectInputStream(fileStream);

Object one = os.readObject();
Object two = os.readObject();
Object three = os.readObject();
GameCharacter elf = (GameCharacter) one;
GameCharacter troll = (GameCharacter) two;
GameCharacter magician = (GameCharacter)
three;
os.close();
```

Downcasting

What happens during de-serialization



Deserialization remarks

- ***Static*** (class) variables **are not serialized** and not restored
- For each ***transient*** instance variable, its constructor is executed
- Objects sent through the network connection can be stamped with a URL which tells where to look for a class definition

Serialization: summary

- You can save the object's state by serializing the object
- You use ObjectOutputStream to serialize an object, which is wrapped around connection stream such as FileOutputStream
- To be serialized, an object must implement Serializable interface
- Mark an instance variable with the transient keyword if you want serialization to skip this variable
- During deserialization the class of all objects in the object graph should be available to JVM
- Static variables are not serialized

The most often used readers

BufferedReader (FileReader)

ObjectInputStream(
BufferedInputStream(
FileInputStream))

The most often used writers

BufferedWriter (FileWriter)

ObjectOutputStream (
    BufferedOutputStream(
        FileOutputStream))

What happens with deserialization if you change your class?

Deserialization fails if:

- You removed an instance variable
- You changed the declared type of instance variable
- You changed a non-transient variable to transient
- You changed an instance variable to static

Changes which do not affect deserialization

- Adding new instance variables
- Changing an access level of instance variable
- Changing a variable from transient to non-transient

In all these cases, new variables will be initialized to their default values

Version UID

- Each time an object gets serialized, it is stamped with a version ID number, generated based on information about the class structure
- If you change your class in any way, its version ID changes, and deserialization fails
- You can put a serial version ID into your serializable class, taking the responsibility that a new version will be compatible with an old one
- To create serial version ID use the serialver tool:

serialver <classname>

The end of Java I/O