Victoria University
of Wellington, New Zealand
Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa

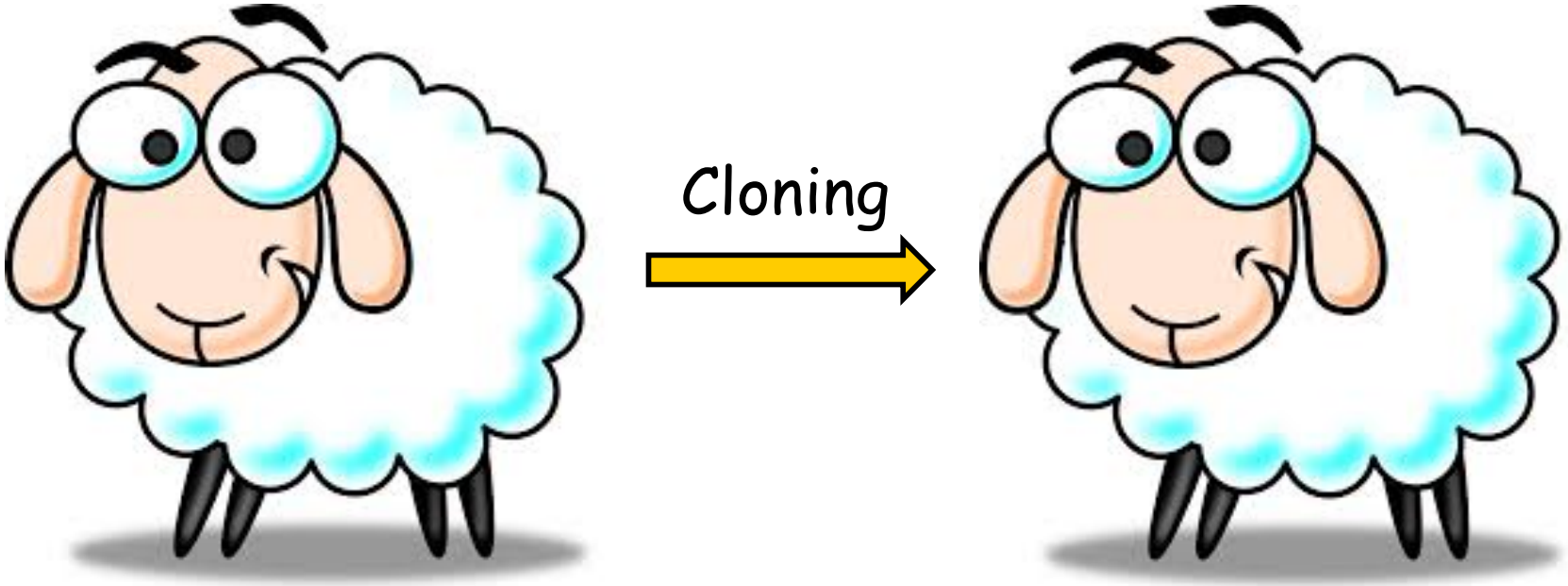
SWEN221:
Software Development

14: Cloning and Serialisation

Why Cloning?

`Sheep dolly`

`dolly.clone()`



Java.lang.Object.clone()

- Purpose is to create copy of object:

```
LispExpr e1 = new LispInteger(1);
LispExpr e2 = e1.clone();
// e1 != e2
// but, e1.equals(e2) must hold and
// e1.getClass() == e2.getClass() must hold
```

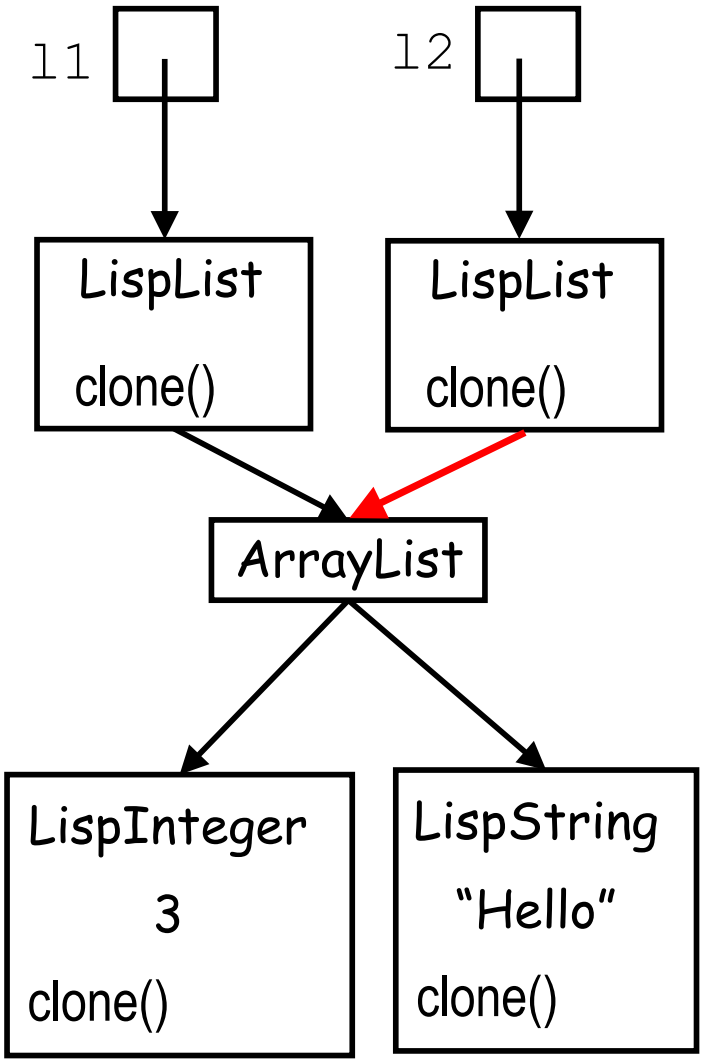
- `Object.clone()` provides default implementation
  - Is **`protected`** so must be explicitly overridden
  - Bitwise copy of all members, including those in subclass

Example clone() implementation

```
class LispList implements Cloneable {
 private List<LispExpr> elements =
                 new ArrayList<LispExpr>();
 …
 public Object clone() {
  try { return super.clone(); }
  catch(CloneNotSupportedException e) {
    return null; // cannot get here
}}}

LispInteger i = new LispInteger(3);
LispString s = new LispString("Hello");
LispList l1 = new LispList();
l1.add(i);
l1.add(s);

LispList l2 = (LispList) l1.clone();
```



- What does this actually do?
  - It performs a **shallow clone**

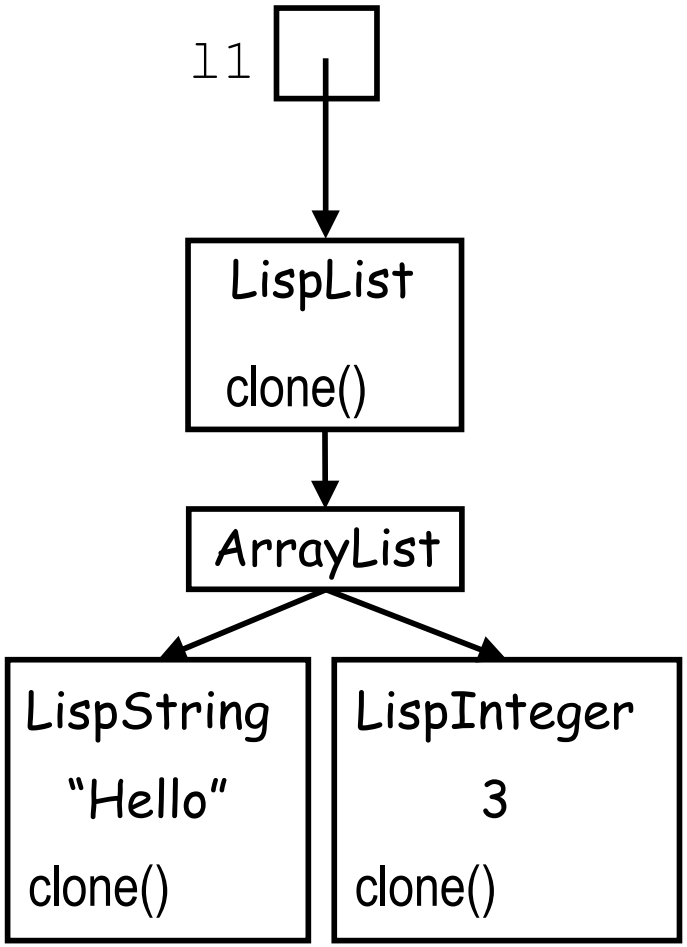
Deep Clone

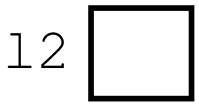
- This version of clone gives a **deep copy**:
  - (i.e. all children recursively cloned)

```
class LispList implements Cloneable {
 private List<LispExpr> elements =
                 new ArrayList<ListExpr>();
 …
 public LispList clone() {
    LispList ne = new LispList();
    for(LispExpr e : elements) {
      ne.add(e.clone());
    }
    return ne;
}}
```

Deep Clone

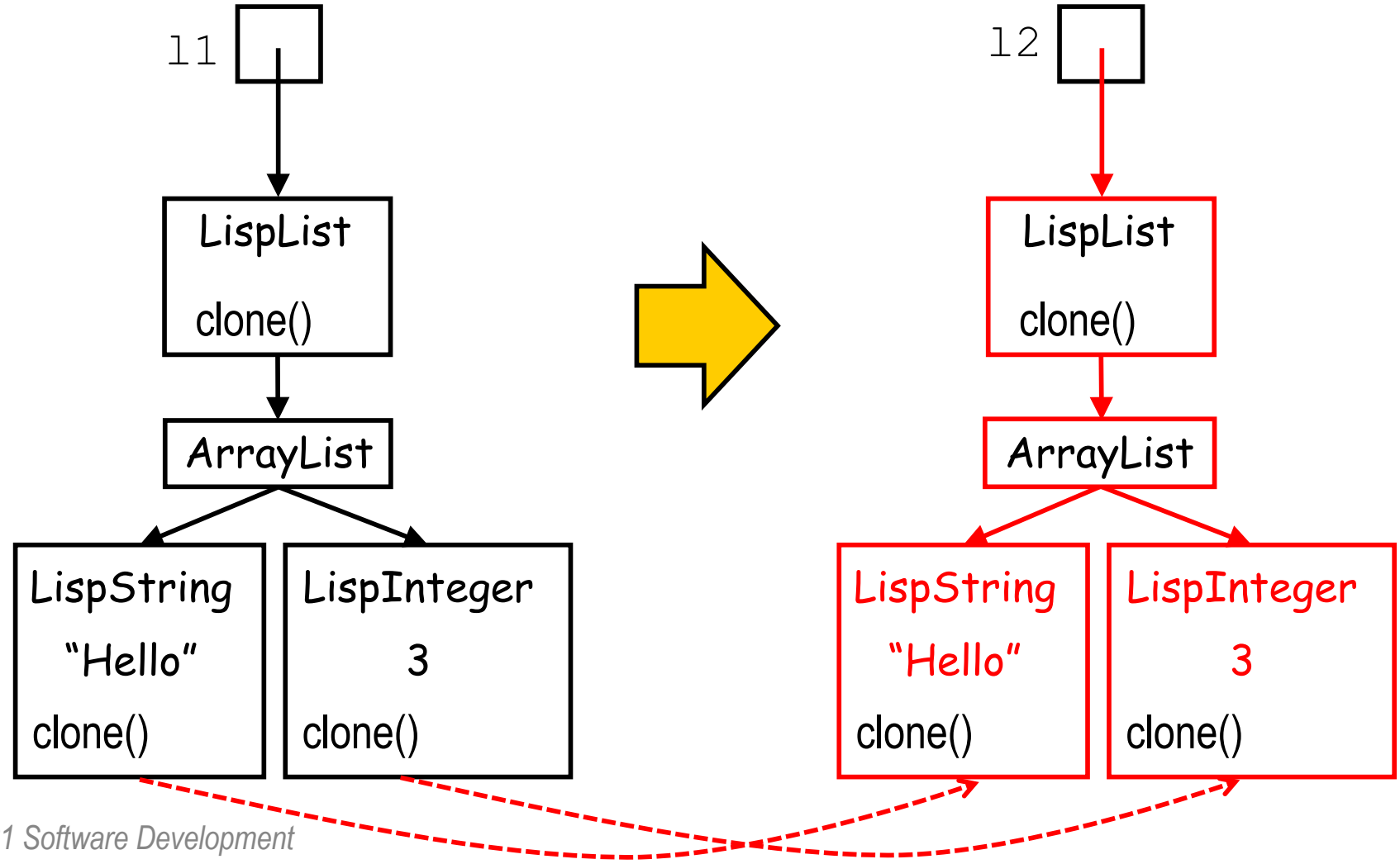
```
l2 = l1.clone();
```





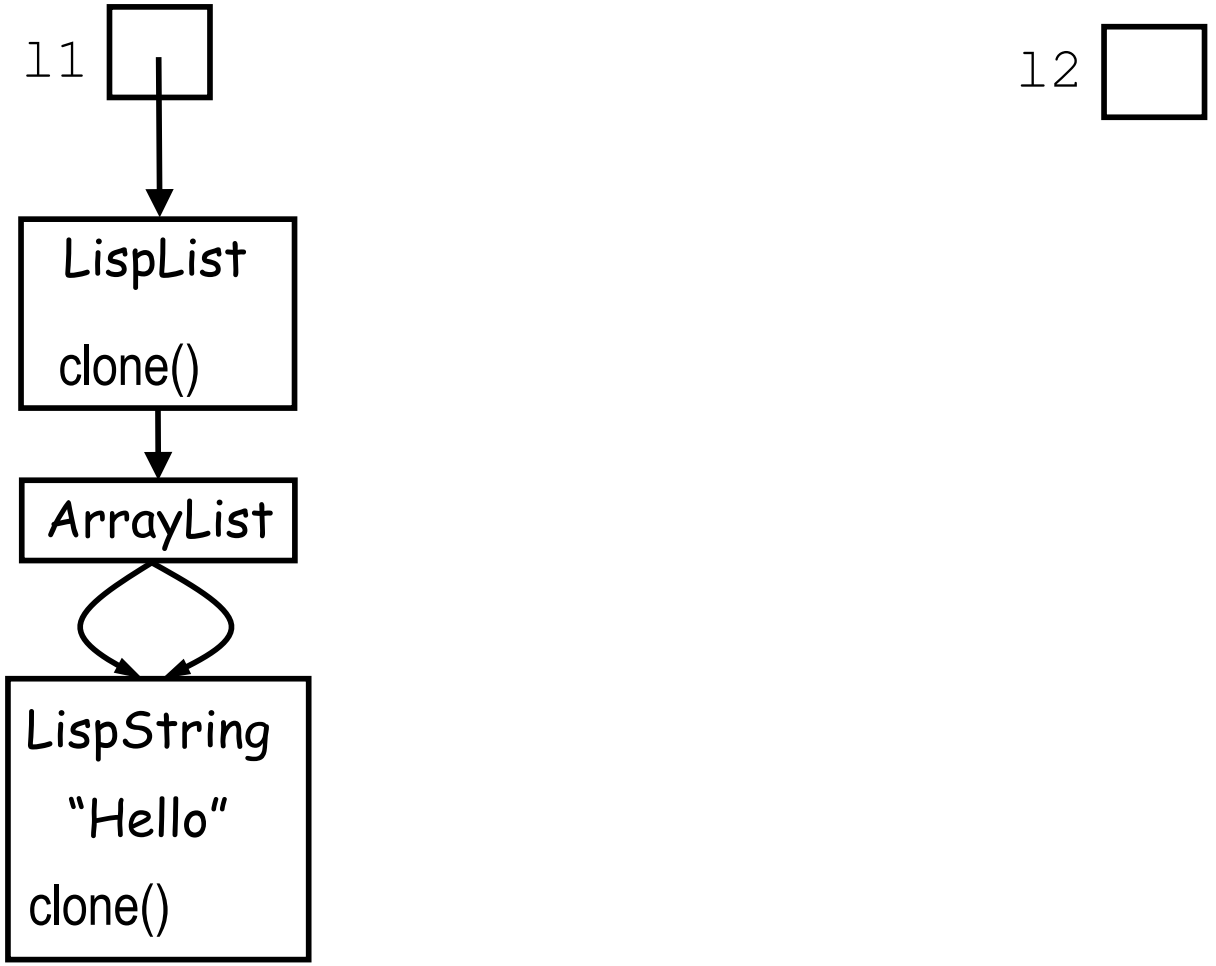
Deep Clone

```
l2 = l1.clone();
```



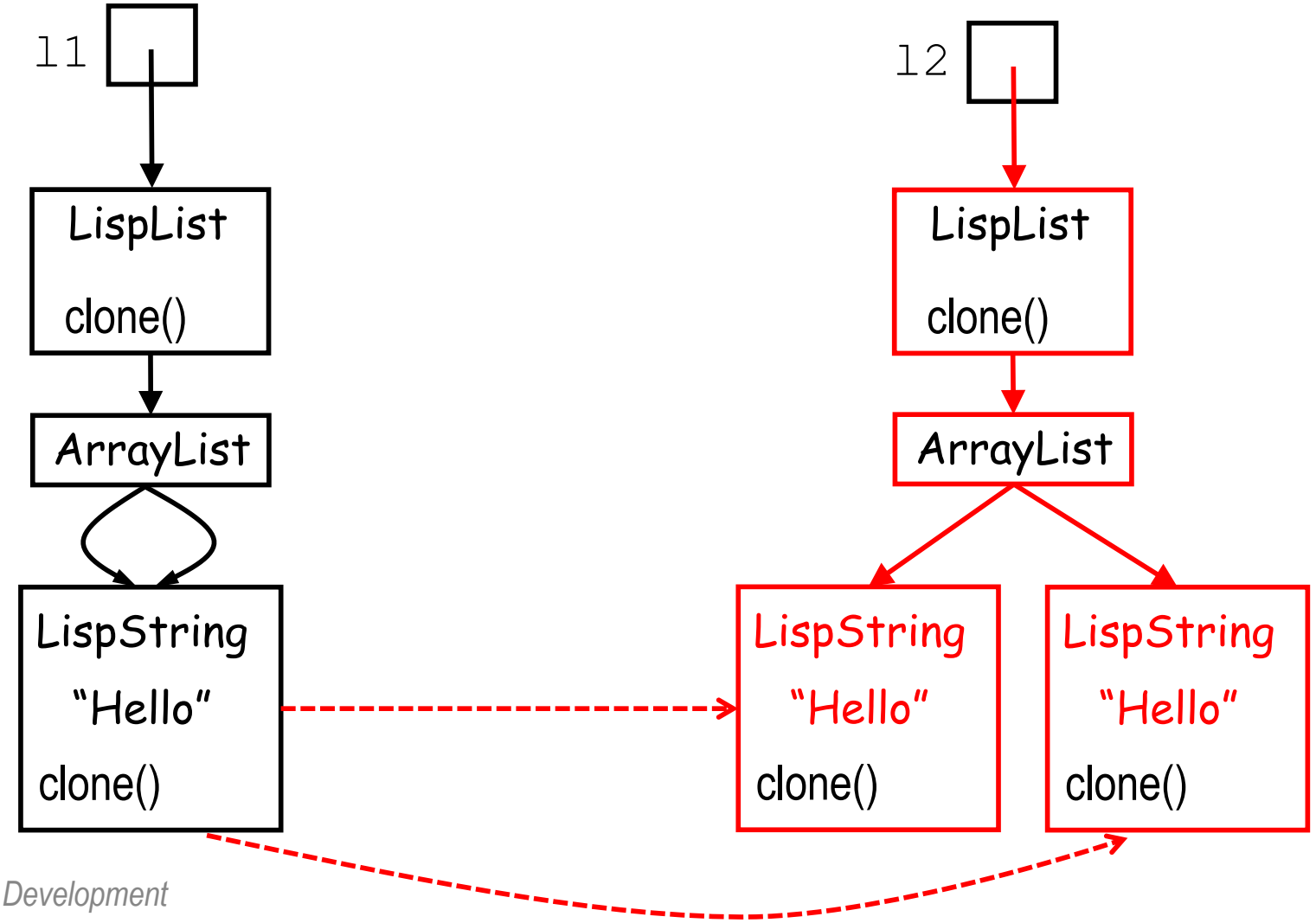
Deep Clone --- What Happens?

```
l2 = l1.clone();
```



Deep Clone --- What Happens?

```
l2 = l1.clone();
```



Few last points on Cloning

- Don’t need to clone immutable types!
  - E.g. Integer, String etc.
  - Why?
- Arrays & Collections
  - clone() is *shallow* – beware!!
- Use super.clone()
- Which to use: deep or shallow copy?
  - Depends upon the situation
  - Always at least clone hidden state
  - One solution is to do both!
    - E.g. by adding a deepClone() method

Quiz: What Will be Printed?

```
class MyClass implements Cloneable {
 public int x;
 public double y;
 public List<String> z;
 public Object clone() {
  try { return super.clone(); }
  catch(CloneNotSupportedException e) { return null; }
}}

List<String> list = new ArrayList<String>();
MyClass c1 = new myClass();
c1.x = 1; c1.y = 1.2; c1.z = list;
MyClass c2 = (MyClass) c1.clone();
c1.x = 2; c1.y = 3.4; c1.z.add("a");

System.out.println(c1.x + "," + c1.y + "," + c1.z.size());
System.out.println(c2.x + "," + c2.y + "," + c2.z.size());
```

Quiz: What Will be Printed?

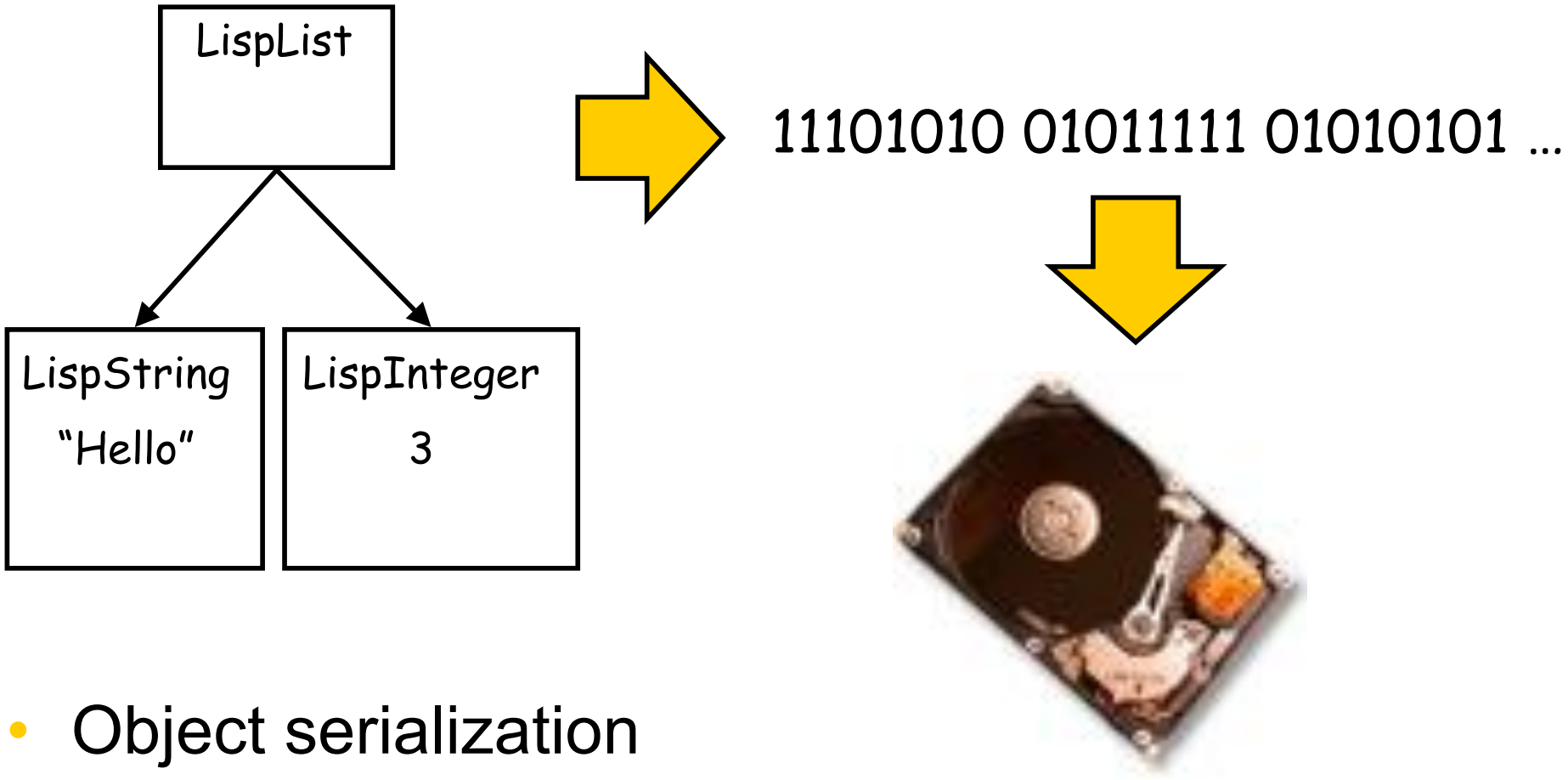
```
class MyClass implements Cloneable {
 public int x;
 public double y;
 public List<String> z;
 public Object clone() {
  try { return super.clone(); }
  catch(CloneNotSupportedException e) { return null; }
}}

List<String> list = new ArrayList<String>();
MyClass c1 = new myClass();
c1.x = 1; c1.y = 1.2; c1.z = list;
MyClass c2 = (MyClass) c1.clone();
c1.x = 2; c1.y = 3.4; c1.z.add("a");

System.out.println(c1.x + "," + c1.y + "," + c1.z.size());
System.out.println(c2.x + "," + c2.y + "," + c2.z.size());
```

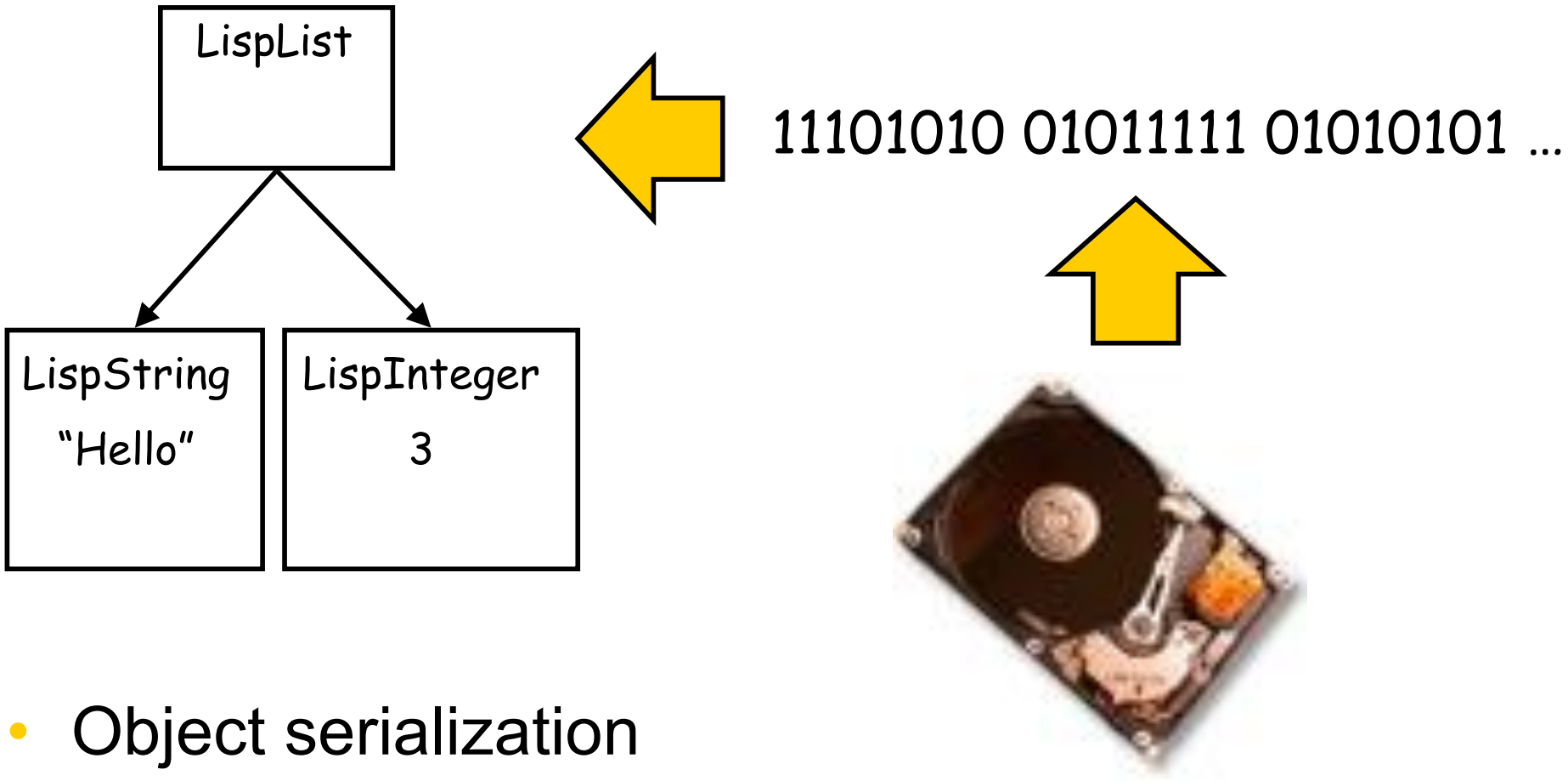
```
2,3.4,1
1,1.2,1
```

Serialisation



- Object serialization
  - Process of converting object into byte sequence
  - Can write sequence to file and…

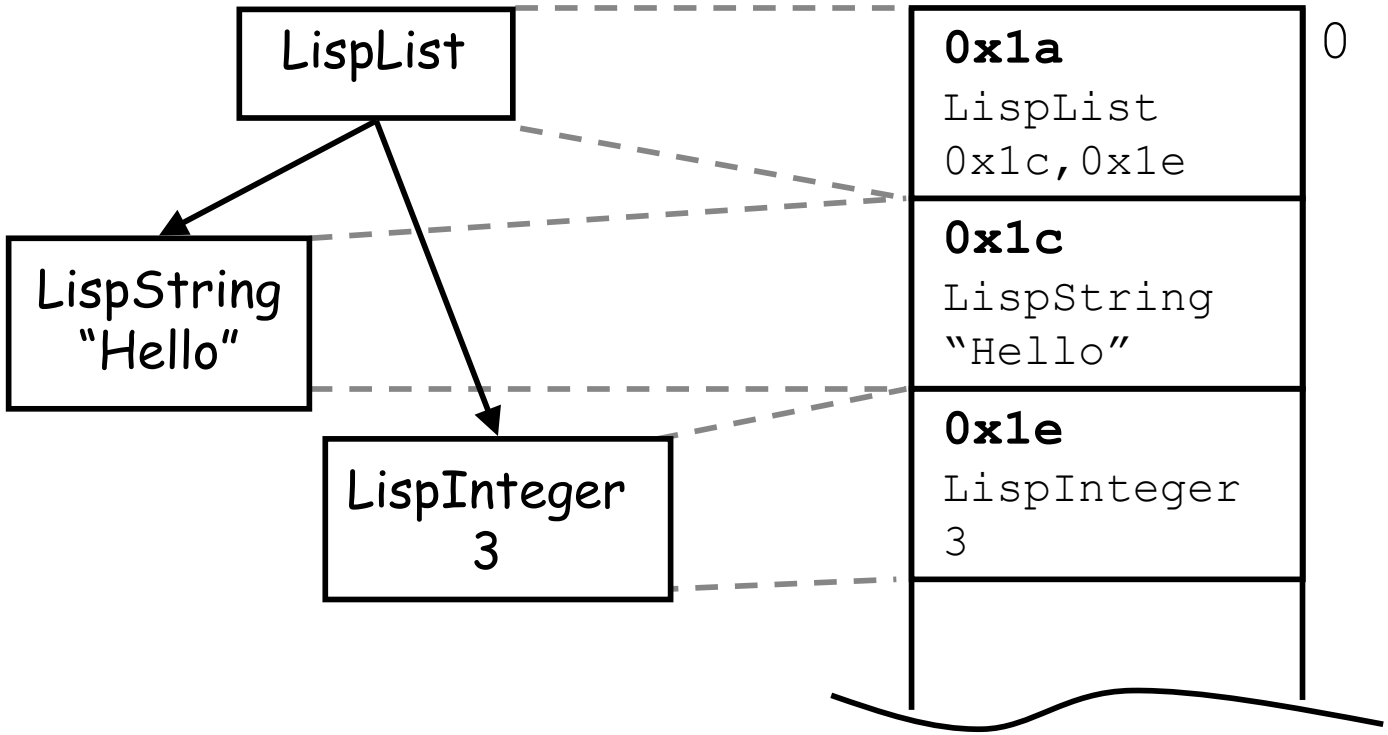
Serialisation



- Object serialization
  - Process of converting object into byte sequence
  - Can write sequence to file and reload it later!

Serialisation

- Consider the following object graph:
  - References are turned into *handles*
  - Primitives (e.g. int) stored in **platform-neutral format**
    - So, can be loaded into machine with different architecture



Serialisation & Deserialisation

```
/** Serialisation into out.ser **/
String obj = "s1";
try {
  FileOutputStream fout = new FileOutputStream("out.ser");
  ObjectOutputStream out = new ObjectOutputStream(fout);
  out.writeObject(obj);
  out.close(); fout.close();
} catch(IOException e) { e.printStackTrace(); }
```

```
/** Deserialisation back from out.ser **/
String s = null;
try {
  FileInputStream fin = new FileInputStream("out.ser");
  ObjectInputStream in = new ObjectInputStream(fin);
  s = (String) in.readObject();
  in.close(); fin.close();
} catch(IOException e) { e.printStackTrace(); }
  catch(ClassNotFoundException e1) {e1.printStackTrace();
}
```

Serialisation & Deserialisation

- **`Class`** `ObjectOutputStream` for serialisation (write objects into stream/file)
  - **`public final void`** `writeObject(Object x)` **`throws IOException`**
  - Standard convention: output to a .ser file

- **`Class`** `ObjectInputStream` for deserialisation (read stream/file into objects)
  - **`public final Object`** `readObject()` **`throws IOException, ClassNotFoundException`**
  - Cast to appropriate data type

Serialisation

- Conditions for serialisation
  - Class must implement `java.io.Serializable` interface
  - All fields in the class must be **serialisable**, or marked **transient** (not to be serialised)
  - Static fields are NOT serialised

- How to check whether a field class is serialisable or not?
  - Look at the document, whether it implements `java.io.Serializable` or not

Example Code

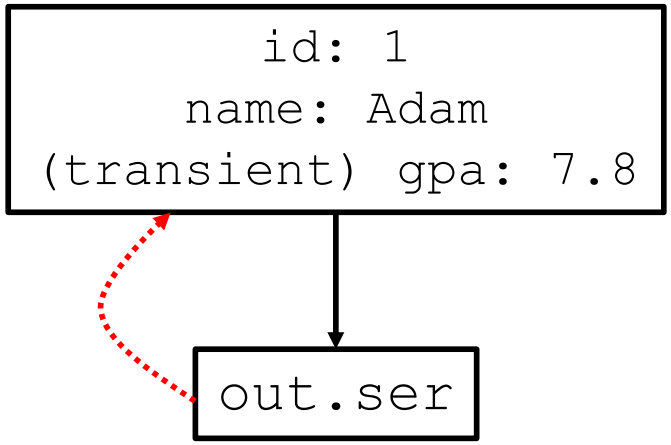
```
public class Student implements java.io.Serializable {
  public String id;
  public String name;
  public transient double gpa;
}

/** Serialisation into out.ser **/
Student s1 = new Student();
s1.id = "1"; s1.name = "Adam"; s1.gpa = 7.8;
try {
  FileOutputStream fout = new FileOutputStream("out.ser");
  ObjectOutputStream out = new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.close();
  fout.close();
} catch(IOException e) {
  e.printStackTrace();
}
```

Example Code

```
/** Deserialisation back from out.ser **/
Student s = null;
try {
  FileInputStream fin = new FileInputStream("out.ser");
  ObjectInputStream in = new ObjectInputStream(fin);
  s = (Student) in.readObject();
  in.close();
  fin.close();
} catch(IOException e) {
  e.printStackTrace();
} catch(ClassNotFoundException e1) {
  e1.printStackTrace();
}

System.out.println("Student id = " + s.id + ", name = " +
s.name + ", gpa = " + s.gpa);
```

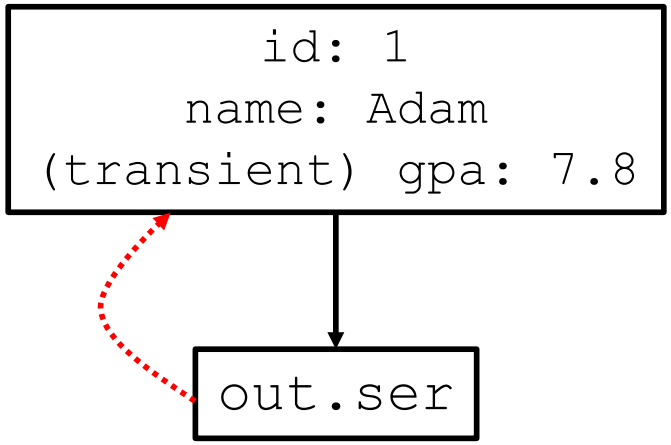


Example Code

```
/** Deserialisation back from out.ser **/
Student s = null;
try {
  FileInputStream fin = new FileInputStream("out.ser");
  ObjectInputStream in = new ObjectInputStream(fin);
  s = (Student) in.readObject();
  in.close();
  fin.close();
} catch(IOException e) {
  e.printStackTrace();
} catch(ClassNotFoundException e1) {
  e1.printStackTrace();
}

System.out.println("Student id = " + s.id + ", name = " +
s.name + ", gpa = " + s.gpa);

Student id = 1, name = Adam, gpa = 0
```



Serialisation Versioning

- Scenario:
  1. `Student s = new Student(); ...`
  2. Write `s` to file "`student.ser`"
  3. Change **`class Student`** (e.g. add field)
  4. Read "`student.ser`" back into program

- Will raise **`InvalidClassException`**!
  - The **version** of the class reading the data is different from the **version** that wrote the data

```
java.io.InvalidClassException: Student; local class
incompatible: stream classdesc serialVersionUID = -
1641098116160791555, local class serialVersionUID =
8075084236435152533
```

Serialisation Versioning

- Versioning
  - Define value for `serialVersionUID`
  - If change **compatible** leave `serialVersionUID` as is
  - If change **incompatible** increment `serialVersionUID`

```
public class Student implements java.io.Serializable {
   public static final long serialVersionUID = 1;
   public String id;
   public String name;
   public transient double gpa;
   …
}

…
```

Serialisation Versioning

- Compatible changes
  - Add fields/classes
  - Change field access (public/private/protected)
  - Change fields from static/transient to non-static/non-transient
  - …
- Incompatible changes
  - Delete fields
  - Change fields from non-static/non-transient to static/transient
  - Change field types (int/double/long/…)
  - Change class position in the class hierarchy
  - Rename class/package
  - …
- *https://docs.oracle.com/javase/8/docs/platform/serialization/spec/version.html*