7 Object serialization

- Object serialization
- Object streams

Well behaved classes

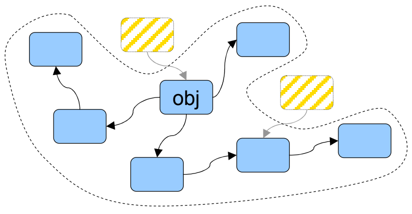
- Objects that are handled by the JVM and many standard classes should have
  - No-arg constructor
  - String representation (toString())
  - Cloning (deep copying)
  - Equality and hashCode methods
  - Serialization (for stream i/o)

Object i/o

- Whole networks of inter-connected objects may be ”flattened” and written to object streams
  - *and later read back into the program again.*
- Typical aplication: Saving the program state for later resumption, e.g. in computer games.

Serialization

- The *object graph* of obj consists of obj and all objects that are directly or indirectly referenced from it.



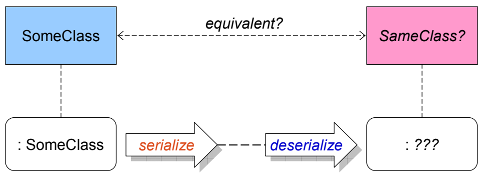
Serialization - deserialization

- When serializing an object a *linear representation* of the graph is built.
- Deserialization means construction of an object graph from a linear representation.
- A class declares that instances may be serialized by implementing

  `interface serializable`

Serialization – deserialization (2)

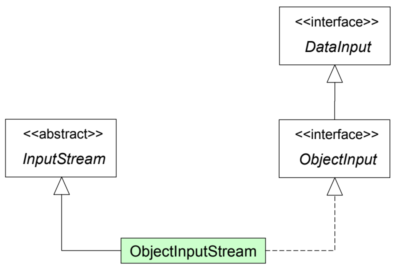
- *How can the runtime environment verify that the class used when deserializing an object is compatible with the class used when serializing it?*



Class version unique identifiers

- The run-time system associates a default `serialVersionUID` with each serializable class.
- In case of mismatch `InvalidClassException` is thrown on deserialization.
- The `serialVersionUID` can (should) be declared explicitly.

Object input stream class relations

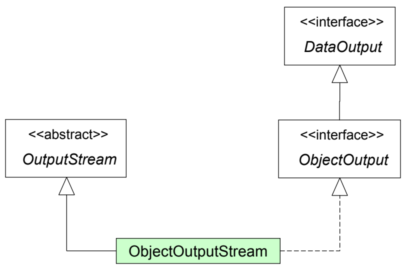


Object input operations

interface ObjectInput
`Object readObject() throws *`
`...`
+ many other methods

`* ClassNotFoundException, IOException`

Object output stream class relations



Object output operations

interface ObjectOutput
`void writeObject() throws *`
`...`
+ many other methods

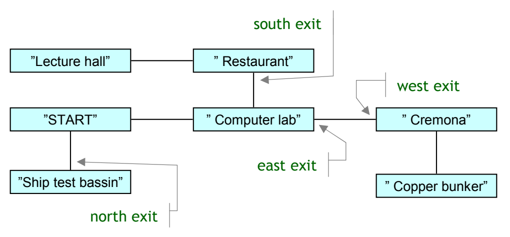
```
* InvalidClassException, NotSerializableException,
  IOException,
```

Example: Adventure game

- Rooms can be created, connected, and explored.
- Rooms contain information.
- Additional information can be added.
- The *room graph* can be saved in a file and reloaded in a future execution.
- Explore the *labyrinth* project.

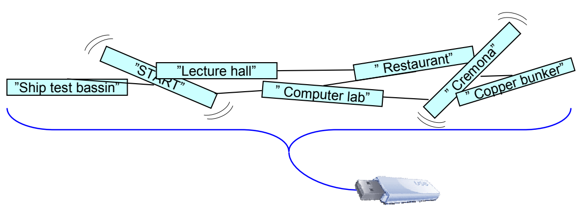
Adventure game (2)

- The noncyclic room graph is a *tree*.
- Room connections are navigable two-ways.



"Graph crushing"

- The room graph is *serialized* and written to an *object stream* connected to a file.



class Room

```
public class Room implements Serializable {
   private String description;
   private HashMap<String,Room> exits;

   public String getInfo() {…}
   public void addInfo(String info) {…}
   public Room getExit(String direction) {…}
   public void connect(String direction,Room room) {…}
}
// Directions: "north","south","east","west"
```

Direction → Room

class Labyrinth

```
public class Labyrinth {
   private static class State implements Serializable {
      Room start = new Room("START");
      Room current = start;
   }
   private State state = new State();

   public void walk(String direction) {…}
   public void addInfo(String comment) {…}
   public void printInfo() {…}
   public void printExits() {…}
   public void save(String fileName) {…}
   public void load(String fileName) {…}
}
```

Room graph root

Relative to current room

Labyrinth.save()

```
public void save(String fileName)
{
   try {
      ObjectOutputStream out =
         new ObjectOutputStream(
            new FileOutputStream(fileName));
      out.writeObject(state);
   }
   catch(Exception e) {
      e.printStackTrace();
      System.exit(0);
   }
}
```

Serialization

Labyrinth.load()

```
public void load(String fileName) {
   try {
      ObjectInputStream in =
         new ObjectInputStream(
            new FileInputStream(fileName));
      state = (State)in.readObject();
   }
   catch(Exception e) {
      e.printStackTrace();
      System.exit(0);
   }
}
```

Deserialization