Serialisering

- En process för att konvertera objekt till en byte-stream som sedan går att återskapa
- Används framförallt som:

  Lagringsteknik: objekt skrivs till fil

  Kopieringsteknik: objekt skrivs till en bytestream i minnet

  Kommunikationsteknik: objekt skrivs till en socket-stream

Interfacet `Serializable`

- Definierar inga metoder - är ett s.k. "Marker" -interface
- Används för att visa att ett objekt av klassen går att serialisera
- Om en superklass implementerar detta interface betyder det att subklassen också går att serialisera (detta går dock att förhindra)

```
public final class User implements Serializable {

 private static final long serialVersionUID = 2376565127898L;
 private String username;
 private String password;

 public User(String username, String password) {
      this.username = username;
      this.password = password;
 }

 public String getPassword() {
      return password;
 }

 public String getUsername() {
      return username;
 }
}
```

Klassen `ObjectOutputStream`

- Används för att serialisera objekt till t.ex. en fil
- Innehåller metoder för att skriva ner objekt samt metoder för att skriva ner andra datatyper (`int`, `float`, `double`, etc.)
- Metoden `writeObject(Object o)` serialiserar ett objekt
- Objektet som skall serialiseras MÅSTE implementera interfacet `Serializable`
- Alla datamedlemmar i objektet som skall serialiseras MÅSTE också implementera `Serializable`

```
User user = new User("Serializer", "oop");
File destination = new File("users.ser");
FileOutputStream fileOutStream = new FileOutputStream(destination);
ObjectOutputStream serializer = new ObjectOutputStream(fileOutStream);

serializer.writeObject(user);
... Close all streams
```

Notera: Bara objektets tillstånd sparas, dvs. objektets datamedlemmar, inte själva objektet

Klassen `ObjectInputStream`

- Används för att läsa objekt som har serialiserats
- Innehåller metoder för att läsa in objekt samt metoder för att läsa in andra datatyper (`int`, `float`, `double`, etc.)
- Metoden `readObject()` läser, deserialiserar, ett objekt
- `readObject()` returnerar objekten som Object-typ och måste därför "castas"
- Värdena måste deserialiseras i den ordning de skrevs till `ObjectOutputStream`

```
File source = new File("users.ser");
FileInputStream fileInStream = new FileInputStream(source);
ObjectInputStream deserializer = new ObjectInputStream(fileInStream);

User user = (User) deserializer.readObject();
... Close all streams
```

Vad sparas och vad sparas inte?

- Alla datamedlemmar som inte är markerade som `transient` sparas
- Alla objekt som icke transient datamedlemmar refererar till sparas också (om dessa inte är `transient`)
- Datamedlemmar som är `static` sparas inte

```
public class Dog implements Serializable {

      private String name; // Sparas
      private Master master; // Sparas
      private transient String type; // Sparas inte
      public static final String SOUND = "Bark!"; // Sparas inte
      ...
}

public class Master implements Serializable {

      private String name; // Sparas
      private transient List dogs; // Sparas inte
      ...
}
```

- När ett Dog-objekt skickas till `writeObject()` kommer ovan angivna datamedlemmar att sparas. Datamedlemmar som är `tranisent` får sina defaultvärden (null för objekt, 0 för int, osv) vid deserialisering.

Vad händer under deserialisering?

- Ett objekt läses från `ObjectInputStream`:
  `User user = (User) objectIn.readObject();`
- JVM:en fastställer vilken klass det är som läses
- JVM:en försöker ladda objektets klass om den inte redan är laddad (hittar den inte klassen kastas ett Exception)
- Det skapas plats på heapen för ett nytt objekt utan att objektets konstruktor anropas
- Om det lästa objektet har en superklass någonstans i sin arvshierarki som inte är `Serializable` anropas dennes default konstruktor följt av dennes superklassers eventuella default konstruktorer
- Objektets datamedlemmar får de värden som serialiserats

Anpassa serialiseringen

- När ett objekt serialiseras sparas alla datamedlemmar som inte är transient eller static
- När ett objekt deserialiseras läses alla datamedlemmar som serialiserats
- Metoderna **`readObject()`** och **`writeObject()`** används för att anpassa serialisering/deserialisering
- Om en serialiserbar klass tillhandahåller dessa metoder skippas den vanliga serialiseringsprocessen

```
// Anropas när objekt av klassen serialiseras
private void writeObject(ObjectOutputStream s) throws IOException {
      s.defaultWriteObject();
      ...
}

// Anropas när objekt av klassen deserialiseras
private void readObject(ObjectInputStream s) throws IOException, ClassNotFoundException {
      s.defaultReadObject();
      ...
}
```

- Metoderna **`defaultReadObject()`** och **`defaultWriteObject()`** kommer att spara och återställa alla datamedelmmar som inte är `transient` eller `static`. Om dessa metoder inte anropas åligger det klassen att tillhandahålla koden för att spara och återskapa dessa datamedlemmar.

Exempel → nästa slide...

Anpassad serialisering

```
public class Dog implements Serializable {

    private String name; // Sparas
    private Master master; // Sparas
    private transient String type; // Sparas inte
    public static final String SOUND = "Bark!"; // Sparas inte

    // Anropas när objekt av klassen serialiseras
    private void writeObject(ObjectOutputStream out) throws IOException {
          out.defaultWriteObject();
          out.writeUTF(type); // sparar värdet från en datamedlem som är transient
    }

    // Anropas när objekt av klassen deserialiseras
    private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException{
          in.defaultReadObject();
          type = in.readUTF(); // återställer en datamedlem som är transient
    }
}
```

Hur förhindras serialisering

- Använd `transient` - nyckelord för att förhindra serialisering av en datamedlem
- Implementera metoderna `readObject()` och `writeObject()` och låt dessa kasta ett `NotSerializableException`

```
public class Dog extends Animal {
	// Anropas när objekt av klassen serialiseras
	private void writeObject(ObjectOutputStream out) throws IOException {
		throw new NotSerializableException(Dog.class.getName());
	}

	// Anropas när objekt av klassen deserialiseras
	private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
		throw new NotSerializableException(Dog.class.getName());
	}
}

public class Animal implements Serializable {
	private static final long serialVersionUID = 20060123L;
}
```

- Animal kommer att gå att serialisera men inte Dog fastän den indirekt implementerar Serializable. Detta för att ett NotSerializableException kastas när writeObject() och readObject() anropas

serialVersionUID

- Används för att versionshantera serialiserade objekt av en klass
- Om ett objekt som serialiserats har ett annat `serialVersionUID` än klassen kommer ett `InvalidClassException` att kastas vi deserialisering
- Varje gång väsentliga ändringar görs av klassen skall `serialVersionUID` ändras (detta gör automatiskt om man inte deklarerar denna datamedlem)

1. User anca01 serialiseras

```
public class User implements Serializable {
      private String username;
}
```

2. Klassen ändras:

```
public class User implements Serializable {
      private String username;
      private String password;
}
```

3. Klassen deserialiseras

4. Ett `InvalidClassException` kastas eftersom objektet som serialiserats inte innehåller datamedlemmen `password`

`serialVersionUID` forts.

- Om `serialVersionUID` tillhandahålls kommer deserialiseringen att fungera

  1. User anca01 serialiseras

```
public class User implements Serializable {
     private static final long serialVersionUID = 22111134423L;
     private String username;
}
```

  2. Klassen ändras:

```
public class User implements Serializable {
     private static final long serialVersionUID = 22111134423L;
     private String username;
     private String password;
}
```

  3. Klassen deserialiseras

  4. Datamedlemmen `password` kommer att få ett defaultvärde (`null` i detta fall)

- **Slutsats:**
  Om ett `serialVersionUID` tillhandahålls tar **du** på dig ansvaret att hantera vad som skall hända när ett äldre objekt deserialiseras. Om klassen har ändrats så att den inte längre är kompatibel med det objekt som tidigare serialiserats bör du ändra `serialVersionUID`.

Att tänka på

- Låt inte ett serialiserbart objekt vara beroende av en `static` datamedlem som kan ändras
- Implementera alltid `equals()` och `hashCode()` i objekt som implementerar `Serializable`
- Deklarera alltid `serialVersionUID` när du implementerar `Serializable`
- Spara inte mer än vad som behövs - implementera `writeObject()` och `readObject()` om så behövs
- Anropa `close()` när du är klar