CS4/MSc Distributed Systems
Tutorial 2: Java programming

Stephen Gilmore
stg@dcs.ed.ac.uk

Tutorial: M 12, Room 5325
Office hour: Th 2, Room 1608

Yussuf Abu-Shaaban
yna@dcs.ed.ac.uk

Tutorial: M 12, Room 6207
Office hour: M 4, Room 3311

Input/output and exceptions

- Java provides a package for input/output, `java.io`, which provides classes for creating input and output streams.
- For efficiency reasons, input and output are usually *buffered*, with data remaining in the buffer until the buffer becomes full, or it is explicitly *flushed* by the application.
- Input and output operations can fail because a file is unreadable, or because we attempt to read past the end of a file. Java signals such failures as *exceptions*. Code which could fail exceptionally is placed in **`try`** .. **`catch`** blocks.

Command line input

```java
import java.io.*;
public class CommandLine {
    private static final InputStreamReader reader =
        new InputStreamReader(System.in);
    private static final BufferedReader stdIn =
        new BufferedReader (reader);
    public static String prompt(String s) {
        try {
            System.out.print(s);
            System.out.flush();
            return stdIn.readLine();
        } catch (IOException ioe) { return null; }
    }
}
```


Using the CommandLine class

```java
class Game {
    public static void main (String[] args) {
        final String theAnswer = "bananas";
        String word = CommandLine.prompt("Guess a word: ");
        while (word != null) {
            if (word.equals(theAnswer)) break;
            word = CommandLine.prompt("Guess again: ");
        }
        if (word != null && word.equals(theAnswer))
            System.out.println("Yes!  You win $1,000,000");
    }
}
```

- The methods `equals()` and `equalsIgnoreCase()` can be used to compare strings.

Control flow and exceptions

```java
class Game2 {
   public static void main(String[] args) {
      class Winner extends Exception {}
      try {
         String word = CommandLine.prompt("Guess a word:");
         while (word != null) {
            if (word.equals("apples"))
               throw new Winner();
            word = CommandLine.prompt("Guess again: ");
         }
      } catch (Winner w) {
         System.out.println("Yes!  You win $1,000,000");
      }
   }
}
```


Data structures

- The *Java Collections Framework* provides data structures which can be used to store related items of data so that they may be accessed efficiently.
- In order that the collections can work with any kinds of data they store them as *objects*. This is termed *object polymorphism*.
- When retrieving data from a collection we must re-instate the class which it had using a *cast*.
- Important kinds of objects include those which implement the `java.lang.Comparable` interface (with a method `compareTo()`).

Common Java programming error

```java
1 import java.util.*;
2 class BadStack {
3     public static void main(String[] args) {
4         Stack s = new Stack();
5         s.push("A string");
6         System.out.println(s.pop().toUpperCase());
7     }
8 }
```

- The `javac` compiler reports an error:

  BadStack.java:6: *cannot resolve symbol*
  symbol: method `toUpperCase()`
  location: class `java.lang.Object`

Casting an object between classes

```java
1 import java.util.*;
2 class GoodStack {
3     public static void main(String[] args) {
4         Stack s = new Stack();
5         s.push("A string");
6         String str = (String)s.pop();
7         System.out.println(str.toUpperCase());
8     }
9 }
```

- The *cast* on line 6 is a *down cast* (or *narrowing*) from `Object` down the class hierarchy to `String`.
- A cast from `String` to `Object` is an *up cast* (or *widening*).

Other data structures

```java
import java.util.*;
class UsingHashtable {
    public static void main(String[] args) {
        Hashtable ht = new Hashtable();
        ht.put("Key1", "Val1");
        ht.put("Key2", "Val2");
        System.out.println(ht.get("Key1")); // prints Val1
    }
}
```

- Hash tables support the methods `put()`, `get()`, `containsKey()`, `containsValue()`, `clear()`, `isEmpty()`, `remove()`, `size()`, and others (see the Java Documentation online for full details). As with stacks, a hash table stores *objects*, so downcasting to the right class is sometimes needed.

Java input and output

- Java provides methods and classes for saving objects to files and recovering them again.
- Saving an object is *serialising* it. Rebuilding it is *deserialising* it.
- Files are manipulated as *streams* of various kinds.
- One type of output stream is an *object stream*.
- Streams are either for input or for output so we have either an *object output stream* or an *object input stream*.

Using an object output stream

```java
import java.io.*;
class UsingObjectOutput {
    public static void main (String[] args)
      throws IOException {
        FileOutputStream fos =
            new FileOutputStream("objectOut.ser");
        ObjectOutputStream os =
            new ObjectOutputStream(fos);
        // serialize String array to the output stream
        os.writeObject(args);
        os.close();
        fos.close();
    }
}
```


Using an object input stream

```java
import java.io.*;
class UsingObjectInput {
    public static void main (String[] args)
      throws IOException, ClassNotFoundException {
        FileInputStream fis =
            new FileInputStream("objectOut.ser");
        ObjectInputStream is = new ObjectInputStream(fis);
        String[] contents = (String[]) is.readObject();
        for (int i = 0; i < contents.length; i++)
            System.out.println(contents[i]);
        is.close();
        fis.close();
    }
}
```