JAVA Cheat Sheet 6
I/O and GUIs

I/O

"A **stream** is a sequence if data."
"A program uses an **input stream** to read data from a source."
"A program uses an **output stream** to write data to a destination." *(docs.oracle.com)*

Byte Streams

In Java all stream types are built upon **ByteStreams**.

```
FileInputStream in
  = new FileInputStream(fileName);
FileOutputStream out
  = new FileOutputStream(fileName);
int c;
while((c = in.read()) != -1){
  out.write(c);
}
```

Character Streams

FileReader and **FileWriter** use *FileInputStream* and *FileOutputStream* internally.

```
FileReader in = new FileReader(fileName);
FileWriter out = new FileWriter(fileName);
int c;
while((c = in.read()) != -1){
 out.write(c);
}
```

Use ints to store last 16 bits instead of last 8 bits.

Line-Oriented I/O

BufferedReader.readLine() uses line terminators to split the lines.

```
BufferedReader in =
  new BufferedReader(new FileReader(fileName));
PrintWriter out =
  new PrintWriter(new FileWriter(fileName));
String line;
while((line = in.readLine()) != null){
 out.println(line);
}
```

Scanner

```
Scanner s = new Scanner(<Stream>);
while(s.hasNext()){
 doSomething...
}
```

Using different delimiter with:

```
s.useDelimiter(Regex);
```

I/O From Command Line

```
InputStreamReader in =
  new InputStreamReader(System.in);
```

Or use **Console**:

```
Console c = System.console();
```

Returns *null* if not available.

Data Streams

Streams to read and write primitive data types.

```
DataInputStream in =
  new DataInputStream(
    new FileInputStream(fileName));

DataOutputStream out =
  new DataOutputStream(
    new BufferedOutputStream(
      new FileOutputStream(fileName)));

in.readDouble(); / out.writeDouble(someDouble);
in.readInt(); / out.writeInt(someInt);
in.readUTF(); / out.writeUTF(someString);
```

Object Streams

Java objects can be written to files if they implement the **Serializable** marker interface.
Every reference inside this object will also be written to the file.
Classes to use are **ObjectInputStream** and **ObjectOutputStream**.

File I/O With NIO

New since Java7.
Class **Path** to represent a path.

```
Path p = Paths.get("/tmp/foo");
```

For releasing resources after use, use **try-with-resources**:

```
try(BufferedWriter writer =
  Files.newBufferedWriter(file, charset));
```

For file handling use class **Files**.

```
Files.write(Path, byte[], OpenOption...);
```

OpenOptions:
{WRITE, APPEND, TRUNCATE_EXISTING, CREATE_NEW, CREATE, DELETE_ON_CLOSE, SPARSE, SYNC, DSYNC}

GUIs

GUIs are event driven. There are **components** which can trigger **events**, and there are **event handlers** that can handle those events.

Swing Components

There are several swing components.

- JLabel
- JTextField
- JButton
- JCheckBox
- JComboBox
- JList
- JPanel

Those components can be added to container objects like JFrame or JPanel.

```
JFrame frame = new JFrame();
frame.add(new JLabel());
frame.add(new JButton());
```

A **layout manager** determines where components are placed within a container.
To make a container or a component visible, the *setVisible()* method has to be called.

```
JFrame frame = new JFrame();
frame.setVisible(true);
```

Event Handling

Different components can create different events. Each event has to be handled by a specific event handler. The event handler has to register to the component in order to handle its events.
There are the following events:

- EventObject
  - EventObject
    - AwtEvent
      - ActionEvent
      - AdjustmentEvent
      - ItemEvent
      - TextEvent
      - ComponentEvent

continued...
→ComponentEvent

- ContainerEvent
- FocusEvent
- PaintEvent
- WindowEvent
- InputEvent
  - KeyEvent
  - MouseEvent
    - MouseWheelEvent

There are also a lot of listeners which all extend EventListener:

- ActionListener
- AdjustmentListener

- ComponentListener
- ContainerListener
- FocusListener
- ItemListener
- KeyListener
- MouseListener
- MouseMotionListener
- TextListener
- WindowListener

To become an event handler a class has to implement the according event listener interface.

```
class ButtonHandler implements ActionListener{
  public void actionPerformed(ActionEvent e){
    System.out.println("Button has been pressed");
  }
}
```

This handler then has to be added to the component.

```
JButton button = new JButton();
button.addActionListener(new ButtonHandler());
```