Agenda

- Recap HashSet, Documentation
- Test-Game
- Main method
- For loop
-

Using sets

```
import java.util.HashSet;
...
HashSet<String> mySet = new HashSet<>();

mySet.add("one");
mySet.add("two");
mySet.add("three");

for(String element : mySet) {
    do something with element
}
```

```
import java.util.ArrayList;
...
ArrayList<String> myList = new ArrayList<>();

myList.add("one");
myList.add("two");
myList.add("three");

for(String element : mySet) {
    do something with element
}
```

Compare with code for an `ArrayList`!

HashSet vs ArrayList

Similarities	Differences:
Both store an arbitrary number of objects	In a HashSet each object can only appear once in the set (because it is a Set). In a ArrayList an Object can appear multiple times.
It is possible to add objects (with the add() method)	An ArrayList is ordered while a HashSet is not ordered.
It is possible to remove objects (with the remove method)	
Both have a size()	
Both provide an iterator() method to go through all the elements	

Tokenising Strings

```
public HashSet<String> getInput()
{
    System.out.print("> ");
    String inputLine =
           reader.nextLine().trim().toLowerCase();

    String[] wordArray = inputLine.split(" ");
    HashSet<String> words = new HashSet<String>();

    for(String word : wordArray) {
        words.add(word);
    }
    return words;
}
```

`List`, **`Map`** and **`Set`**

- Alternative ways to group objects.
- Varying implementations available:
  - **`ArrayList`**, **`LinkedList`**
  - **`HashSet`**, **`TreeSet`**
- But **`HashMap`** is unrelated to **`HashSet`**, despite similar names.
- The second word reveals organizational relatedness.

Writing class documentation

Your own classes should be documented the same way library classes are.

Other people should be able to use your class without reading the implementation.

Make your class a potential 'library class'!

Elements of documentation

Documentation for a class should include:

- the class name
- a comment describing the overall purpose and characteristics of the class
- a version number
- the authors' names
- documentation for each constructor and each method

Elements of documentation

The documentation for each constructor and method should include:

- the name of the method
- the return type
- the parameter names and types
- a description of the purpose and function of the method
- a description of each parameter
- a description of the value returned

javadoc

Class comment:

```
/**
 * The Responder class represents a response
 * generator object. It is used to generate an
 * automatic response.
 *
 * @author     Michael Kölling and David J. Barnes
 * @version    1.0  (2016.02.29)
 */
```

javadoc

Method comment:

```
/**
 * Read a line of text from standard input (the text
 * terminal), and return it as a set of words.
 *
 * @param  prompt  A prompt to print to screen.
 * @return A set of strings, where each String is
 *         one of the words typed by the user
 */
public HashSet<String> getInput(String prompt)
{
   ...
}
```

Public vs private access modifier (Zugriffsmodifikator)

Public elements are accessible to objects of other classes:

- Fields, constructors and methods
- Only methods that are intended for other classes should be public.

Fields should not be public.

Private elements are accessible only to objects of the same class.

- to break up a large task into several smaller ones
- Subtask needed for several methods of a class

Protected will be discussed later

Information hiding

Data belonging to one object is hidden from other objects.

Know *what* an object can do, not *how* it does it.

Information hiding increases the level of *independence*.

Independence of modules is important for large systems and maintenance.

Review

Java has an extensive class library.

A good programmer must be familiar with the library.

The documentation tells us what we need to know to use a class (interface).

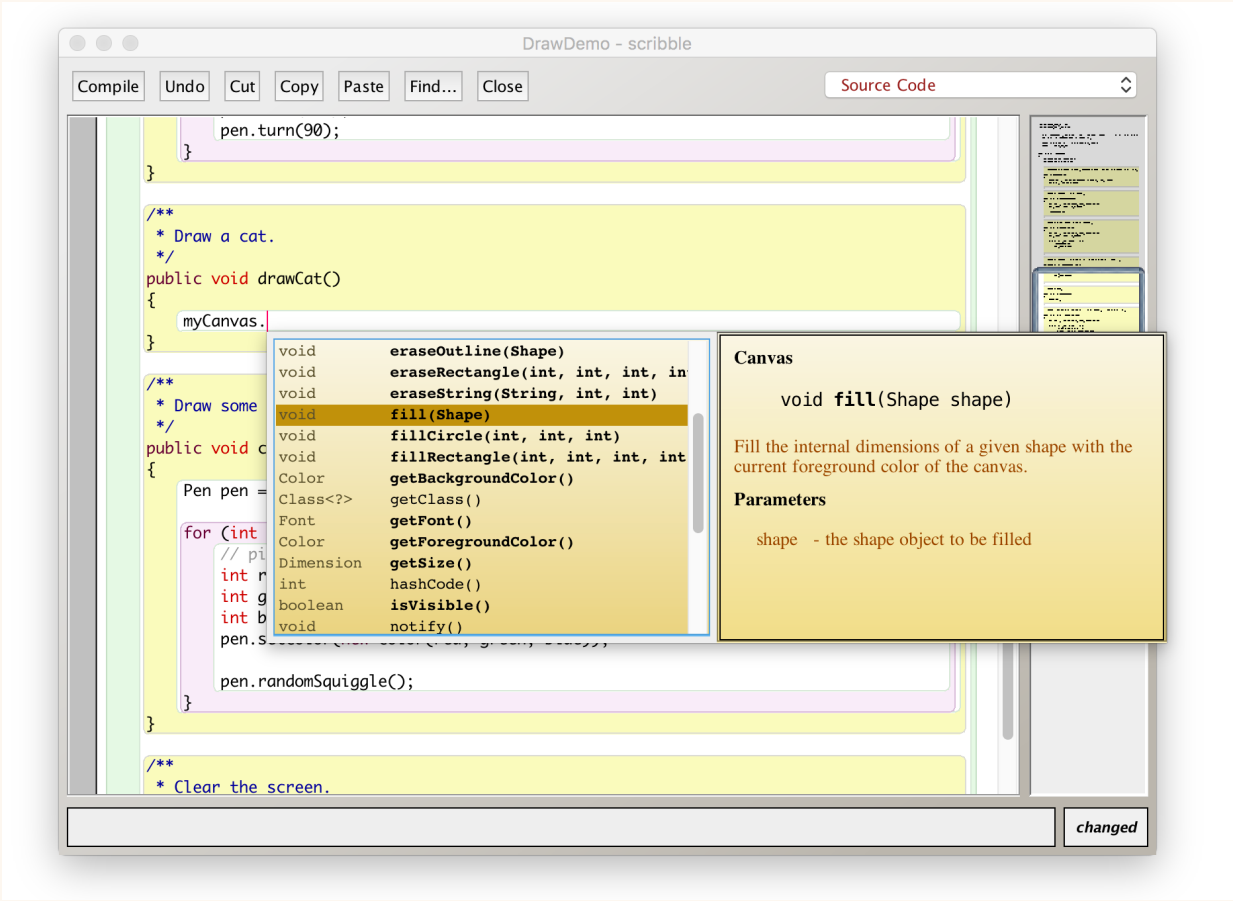
The implementation is hidden (information hiding).

We document our classes so that the interface can be read on its own (class comment, method comments).

Code completion

- The BlueJ editor supports lookup of methods.
- Use Ctrl-space after a method-call dot to bring up a list of available methods.
- Use *Return* to select a highlighted method.

Code completion in BlueJ



Main method

https://www.youtube.com/watch?v=cSlCoWJ4CBc

Our programs are dependent upon at least one instantiation (object) of a class within the BlueJ environment. We have not yet coded a way to run our programs outside the BlueJ environment.

There is a method called main which is the defined entry point for starting programs in Java, and can be used to create objects of other classes and call their methods.

```
public static void main(String[] args)
```

Loops: repeated execution of a sequence of statements

for-each loop

```
for(String filename : files) {
        System.out.println(filename);

}
```

while loop

```
int index = 0;
while(index < files.size()){
      System.out.println(filename);
      index++;
}
```

The for loop

There are two variations of the for loop, *for-each* and *for*.

for-each was introduced very late to java (5) -> no new Keyword!

The for loop is often used to iterate a fixed number of times.

Often used with a variable that changes a fixed amount on each iteration.

A Java example

Initialization
Condition
Update
block is executed repeatedly

for loop version

```
for(int index = 0; index < 10; index++) {
    notes.add("notes"+index);
}
```

while loop version

```
int index = 0;
while(index < 10) {
    notes.add("notes"+index);
    index++;
}
```

For loop pseudo-code

General form of the for loop

```
for(initialization; condition; post-body action) {
    statements to be repeated
}
```

Equivalent in while-loop form

```
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```

for loop with bigger step

```
// Print multiples of 3 that are below 40.
  for(int num = 3; num < 40; num = num + 3)
  {
      System.out.println(num);
  }
```