Classes and objects, part 1

01

- Creating objects
- Instance methods and fields
- Object identity and equality
- Memory management

Objects

- Java is an *object-oriented* language.
- Objects combine state (data) and code that operates on the object state.
  - State: object fields (variables).
  - Code: contained in methods.
- Example: a `String` object stores the character sequence (data), and has many methods, e.g., `char charAt(int index)`.

Classes

- The type of an object is a class.
- The class is a “template” for objects of the type, defining
  - the data stored (fields and their type); and
  - the methods of the object.
- Java allows the programmer to define new classes, so the set of object types is unlimited.

Creating objects: the `new` operator

- Create a new object:

```
String s0 = new String();
char[] chArray = {'H', 'e', 'l', 'p'};
String s1 = new String(chArray);
```

  - keyword `new`; followed by
  - class name (here, `String`); followed by
  - constructor arguments, in parenthesis.
    - What constructor arguments depends on the class; can have several constructors.
- Object creation is an *expression*: it evaluates to the new object.

Instance methods

- (non-static) Instance methods are called on an object, and can access (depend on, and modify) that object, as well as their arguments.
- Instance method call:

```
char c = s1.charAt(s1.length() - 1);
```

  - an expression that yields an object;
  - a dot (“.”);
  - the name of the method, followed by arguments in parentheses.

Example: String functions

- Use `String`'s methods to:
  - Count words in a string.
- Use `StringBuffer`'s methods to:
  - Turn an `int[]` into a (readable) `String`.
  - Replace all occurrences of a word in a string.

Arrays are objects

- Create a new array:

```
int[] = new int[2 * n];
```

  - The array is initialised with the *default value* for its element type (0 for primitive numeric types).
- `length` is a (read-only) property of the array object.

Object identity & equality

- Variables of primitive type hold a value of the type, while variables of a class type hold a reference to an object of the class.
  - Two variables can refer to the same object.
  - Two objects can contain the same data.
- A class type variable (expression) can take the value `null`, meaning “no object”.
- For primitive types, `a == b` iff `a` equals `b`.
- For class types,
  - `a == b` iff `a` and `b` are *the same* object;
  - `a.equals(b)` iff `a` and `b` are *equal* (class-dependent definition).

```
String s = "hello";
char[] chArray = {'h', 'e', 'l', 'l', 'o'};
String t = new String(chArray);
System.out.println("s = " + s);
System.out.println("t = " + t);
System.out.println("s == t ? " + (s == t));
System.out.println("s.equals(t) ? " + s.equals(t));

s = null;
// two of these cause runtime exception:
System.out.println("s.length = " + s.length());
System.out.println("s.equals(t) ? " + s.equals(t));
System.out.println("t.equals(s) ? " + t.equals(s));
```

The heap and garbage collection

- Dynamically created (`new`) objects are stored in a memory called the heap.
- In Java, we do not have to (indeed cannot) explicitly delete objects.
- A garbage collector automatically reclaims heap space used by objects that are no longer referenced (directly or indirectly) by the program.
  - Can we still run out of memory? Yes.