# Everything Is an Object

### Where Storage Lives

It’s useful to visualize the way things are laid out while the program is running—in particular, how memory is arranged. There are five different places to store data:

1. **Registers**. This is the fastest storage because it exists in a place different from that of other storage: inside the processor. However, the number of registers is severely limited, so registers are allocated as they are needed. You don’t have direct control, nor do you see any evidence in your programs that registers even exist (C & C++, on the other hand, allow you to suggest register allocation to the compiler).
2. **The stack**. This lives in the general random-access memory (RAM) area, but has direct support from the processor via its *stack pointer*. The stack pointer is moved down to create new memory and moved up to release that memory. This is an extremely fast and efficient way to allocate storage, second only to registers. The Java system must know, while it is creating the program, the exact lifetime of all the items that are stored on the stack. This constraint places limits on the flexibility of your programs, so while some Java storage exists on the stack—in particular, object references—Java objects themselves are not placed on the stack.
3. **The heap**. This is a general-purpose pool of memory (also in the RAM area) where all Java objects live. The nice thing about the heap is that, unlike the stack, the compiler doesn’t need to know how long objects must stay on the heap. Thus, there’s a great deal of flexibility when using heap storage. Whenever you need an object, you write the code to create it using **new**,and the storage is allocated on the heap when that code is executed. There’s a price for this flexibility: It can take more time to allocate and clean up heap storage than stack storage (if you even *could* create objects on the stack in Java, as you can in C++).
4. **Constant storage**. Constant values are often placed directly in the program code, which is safe since they can never change. Sometimes constants are cordoned off by themselves so they can be optionally placed in read‑only memory (ROM), in embedded systems.[[1]](#footnote-1)
5. **Non-RAM storage**. If data lives completely outside a program, it can exist while the program is not running, outside the control of the program. The two primary examples of this are *streamed objects,* in which objects are turned into streams of bytes, generally sent to another machine, and *persistent objects,* in which the objects are placed on disk so they will hold their state even when the program is terminated. The trick with these types of storage is turning the objects into something that exist on the other medium, and yet be resurrected into a regular RAM-based object when necessary. Java provides support for *lightweight persistence*, and mechanisms such as JDBC and Hibernate provide more sophisticated support for storing and retrieving object information in databases.

# Holding Your Objects

## **List**

**List**s promise to maintain elements in a particular sequence. The **List** interface adds a number of methods to **Collection** that allow insertion and removal of elements in the middle of a **List**.

There are two types of **List**:

* The basic **ArrayList**, which excels at randomly accessing elements, but is slower when inserting and removing elements in the middle of a **List**.
* The **LinkedList**, which provides optimal sequential access, with inexpensive insertions and deletions from the middle of the **List**. A **LinkedList** is relatively slow for random access, but it has a larger feature set than the **ArrayList**.

The following example reaches forward in the book to use a library from the *Type Information* chapter by importing **typeinfo.pets**. This is a library that contains a hierarchy of **Pet** classes along with some tools to randomly generate **Pet** objects.

1. An example of this is the string pool. All literal strings and string-valued constant expressions are interned automatically and put into special static storage. [↑](#footnote-ref-1)