Review of objects and variables in Java

Variables and Objects
- what happens when you run this?
  String a = “foo”;
  System.out.println (a);
  - it prints
    foo
- what is “foo”?
  - a string literal that evaluates to a String object
  - what is a?
  - a variable whose value is an object reference
  - what is String a = “foo”?  
    - a declaration and an assignment in one

Method Calls
- what happens when you run this?
  String a = “foo”;
  String b = a.toUpperCase ();
  System.out.println (a);
  - it prints
    foo
- what is toUpperCase?  
  - a method of class String
    type is String -> String
    declared as public String toUpperCase ()
- what is a.toUpperCase ()?
  - a method call on the object a
  - does it change a?  
    no, it creates a new string

Null References
- what happens when you run this?
  String a = null;
  System.out.println (a);
  - it prints
    null
- what happens when you run this?
  String a = null;
  String b = a.toUpperCase ();
  System.out.println (b);
  - it throws a NullPointerException
  - why?
    because a method call must have a subject
Sharing & Equality

– what happens when you run this?
  String a = "foo";
  String b = "foo";
  System.out.println (b);
– it prints
  foo
– is that the same as this?
  String a = "foo";
  String b = a;
  System.out.println (b);
– yes, because String is immutable.
– There is no way to distinguish these cases and, in fact, Java virtual machine may produce upper or lower state in this case.

Mutable Containers

– what happens when you run this?
  Vector v = new Vector ();
  String a = "foo";
  v.addElement (a);System.out.println (v.lastElement ());
– it prints
  foo
– what happens when you run this?
  Vector v = new Vector ();
  String a = "foo";
  String b = "foo";
  v.addElement (a);
  System.out.println (v.lastElement ());
  v.addElement (b);
  System.out.println (v.lastElement ());
– it prints
  foo
  foo

Aliasing

– what about this?
  Vector v = new Vector ();
  Vector q = v;
  String a = "foo";
  v.addElement (a);
  System.out.println (q.lastElement ());
– it prints
  foo
– why?
– because v and q are aliased: they are names for the same object
– what if we now do this?
  if (v == q) System.out.println ("same object");
  if (v.equals (q)) System.out.println ("same value");
– it prints
  same object
  same value

Aliasing occurs when several different identifiers refer to the same object. The term is very general and is used in many contexts.

Aliasing and Immutables

– what does this do?
  String a = "foo";
  String b = a;
  a.toUpperCase ();
  System.out.println (b);
it prints
  foo
– why?
  because strings are immutable
– The objects created by the toUpperCase method is eventually GCed (garbage collected.)
Polymorphism

• what does this do?
  Vector v = new Vector();
  Vector e = new Vector();
  v.addElement (e);
  e.addElement ("foo");
  System.out.println (
    ((Vector) v.lastElement ()).lastElement ()�);
• it prints
  foo
• what kind of method is addElement?
  a polymorphic one
type is Vector, Object -> void
declared as public void addElement (Object o)

On Polymorphism

• First identified by Christopher Strachey (1967) and
developed by Hindley and Milner, allowing types such as a
list of anything.
• E.g. in Haskell we can define a function which operates on
a list of objects of any type a (a is a type variable).
  length :: [a] -> Int
• Polymorphic typing allows strong type checking as well as
generic functions. ML in 1976 was the first language with
polymorphic typing.
• Ad-hoc polymorphism (aka overloading) is the ability to
use the same syntax for objects of different types, e.g. "a + b"
for addition of reals and integers.
• In OOP, the term is used to describe variables which may
refer at run-time to objects of different classes.

Reference Loops

• Can one even add v to itself?
  Vector v = new Vector();
  v.addElement (v);
  System.out.println (v.lastElement ());  
• yes, try it!
• and this?
  v.addElement (5);
• no, 5 is a primitive value, not an object

A Pair of Methods

• Some types
  – what are the types of addElement, lastElement?
    addElement : Vector, Object -> void
    lastElement : Vector -> Object
• A puzzle
  – how are x and e related after this?
    v.addElement (e);
    x = v.lastElement ();
  – they denote the same object
  – can the compiler infer that?
  – no! not even that x and e have the same class
Downcasts

• What does this do?
  Vector v = new Vector ();
  String a = “foo”;
  v.addElement (a);
  String b = v.lastElement ();
  System.out.println (b);

• Compiler rejects it: v.lastElement doesn’t return a String!

• what does this do?
  Vector v = new Vector ();
  String a = “foo”;
  v.addElement (a);
  String b = (String) v.lastElement ();
  System.out.println (b);

• it prints
  foo

Upcasting and Downcasting

• Suppose we have object O of class C1 with superclass C2
• In Java, upcasting is automatic but downcasting must be explicit.
• Upcasting: treating O as a C2
• Downcasting: treating O as a C1

Variable & Object Classes

• What does this do?
  Vector v = new Vector ();
  String a = “foo”;
  v.addElement (a);
  Object o = v.lastElement ();
  System.out.println (o.getClass ());

• It prints
  java.lang.String

• What’s going on here?
  – getClass returns an object representing a class
  – o.getClass () is the class o has at runtime
  – System.out.println prints a string representation, i.e., the name

Some Key Concepts

• Variables & objects
  – variables hold object references (or primitive values like 5)
  – null is a special object reference

• Sharing, equality & mutability
  – distinct objects can have the same value
  – state is held in value of instance variables
  – an object can be mutable (state may change) or immutable
  – two variables can point to the same object; changing one affects the other

• Methods
  – a method has a ‘subject’ or ‘target’ object
  – may be polymorphic, i.e., work on several types of object

• Compile-time & runtime types
  – an object has a type at runtime: the class of its constructor
  – a variable has a declared, compile-time type or class
  – runtime class is subclass of compile-time class