

Introduction to Java

Tailored for GUI Programming

Outline

- Program structure
- Variables and objects
- Type checking
- Dispatching

Program structure

- **Classes and interfaces**
 - made of methods and fields
 - static versus instance methods & fields
 - nested classes, anonymous classes
- **Packages**
 - naming. fully-qualified names
 - encapsulation via access control
- **Access control**
 - for classes and interfaces
 - for class members

Variables and objects

- Variable = name + storage (value)
- 2 kinds of value
primitives & references to objects
- Local vars are stored on stack, and objects are stored on heap.
- == versus r.equals(o)
- Object: creation, sharing, mutability
- Method invocation & activation record

Variables and objects: ex

```
int i=6;
int j; // uninitialized
int [] a ={1,3,5,7,9}; // create a 5-element array
int [] b = new int[3];
String s = "abcdef"; // create a new string
String t = null;
```

Sharing and mutability

```
int i=6;
int j; // uninitialized
int [] a = {1,3,5,7,9}; // create a 5-element array
int [] b = new int[3];
String s = "abcdef"; // create a new string
String t = null;
```

```
j = i;
b = a;
```

```
t = t+"g";
b[0] = i+1;
a[1] = (a[0]==7)?5:3;
```

Method invocation

```
class Arrays {
    public static void multiplies(int[] a, int m){
        if (null==a) return;
        for (int i=0; i<a.length; ++i){
            a[i] = a[i]*m;}
        }
}

int [] b = {1, 3, 5, 7, 9};
Arrays.multiplies(b, 2); // pass by value
```

Types

- types = primitive types + object types
- Object types defined by classes and interfaces.
- Subtyping relation. *extends* versus *implements*
- A supertype captures common properties of subtypes. e.g., all classes are subtypes of `java.lang.Object`.
 - Object defines `equals()`, `hashCode()`, `toString()`, etc
- Assignment rule:
 $l=r; T(l) \geq T(r)$

Type checking

- Type checking rules out *type errors*.
- But, type checking cannot rule out all *programming errors*.
 - “(struct S *) (0x1) -> m” compiles in C
- Type errors are only a subset of programming errors.

Declaration types, apparent types, and actual types

```
int x, y;
```

```
static int gcd(int m, int n);
```

```
int s = gcd(x,y);
```

```
long t = gcd(x,y);
```

```
String s = "abc";
```

```
Object o = s;
```

Dispatching

```
String t="ab";  
Object o = t+"c";  
Object r = "abc";  
boolean b = o.equals(r);
```

What is the value of b and why?

Dispatching

- When a method is called on an object, it is essential to invoke the code provided by the object.
- Dispatch vector:
a runtime mechanism that stores list of methods defined by object's class.
- Object contains a reference to the vector.

Garbage collection

An object's memory is reclaimed by the virtual machine automatically if there are no variables referring to the object.

```
class Stack{  
    Object[] data;  
    int p;  
    Stack(int size) {data = new Object[size];p=0;}  
    void push(Object e)  
    {data[p++]=e;} // may throw IndexOutOfBoundsException  
    Object pop() {return data[--p];}...  
}
```

Conversions and overloading

- Explicit conversions, a.k.a casting
(T)d
- Implicit conversions (widening)
 - e.g. int->long, float->long
- Overloading
 - operator overloading, e.g., +
 - method name overloading

Overloading resolution: ex 1

“Most-specific” rule:

- methods requiring least conversions win
- tie as compilation error

```
static int comp(int,long) // def 1
static int comp(long,int) // def 2
static int comp(long,long) // def 3

int x; long y; float z;

C.comp(x,y)
C.comp(z,z)
C.comp(x,x)
```

Overloading resolution: ex2

```
static void comp(Sup,int)    // def 1
```

```
static void comp(Sub,long)  // def 2
```

Sup is supertype of Sub

```
Sub e; int i;
```

```
C.comp(e,i);
```

Exercise 1

```
Object o = "abc";  
boolean b = o.equals("a, b, c");  
char c = o.charAt(1);  
Object o2 = b;  
String s = o;  
String t = (String)o;  
c=t.charAt(1);  
c=t.charAt(3);
```

Exercise 2

```
char[] a = {'a','b','c'};
Object o = "abc";
String t="ab";
String w = t+"c";
String u=w;

boolean b0 = o.equals(a);
boolean b1 = o.equals(t);
boolean b2 = o.equals(w);
boolean b3 = (o==w);
boolean b4 = (u==w);
```

Exercise 3

```
void m(Object o, long x, long y) // def 1
void m(Object o, int x, long y)  // def 2
void m(String s, int x, long y)  // def 3
void m(String s, long x, int y)  // def 4

Object u; String v; int a; long b;

m(v, a, b);
m(v, a, a);
m(v, b, a);
m(v, b, b);
```

Summary

- Program structure
- Variables and objects
- Type checking
- Dispatching