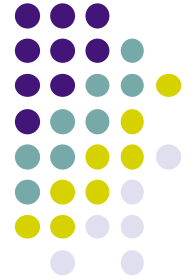


Static typing versus dynamic typing



- A major property of a language is whether it is statically or dynamically typed
- **Static typing**: Variable types are known at compile time
 - Java, Scala, Haskell
- **Dynamic typing**: Variable types are not known at compile time but only at run time
 - Ruby, Python, Erlang, Scheme, Oz (language of this course)
- Static typing versus dynamic typing?
 - This question evokes intense debate between language designers
 - The main issues are **guarantees and flexibility**
 - Java augments static typing with concepts to increase flexibility
 - An Object class that is the root of the class hierarchy
 - The ability to define class code at run time with a class loader



Types in Java

- Two kinds of types: primitive types and reference types
 - User-defined types (e.g., classes) are reference types
- **Primitive type**: boolean (1 bit), character (16 bits), byte (8 bit integer, -128..127), short (16), int (32), long (64), float (32), double (64)
 - **Characters**: Unicode standard (all written languages)
 - **Integers**: representation in 2's complement
 - **Floating point**: IEEE754 standard
- **Reference type**: class, interface, or array
 - A value is either “null” or a reference to an object or an array
 - An array type has the form $\tau[]$ where τ can be any type

Object-oriented programming in Java



- Data abstraction in Java
 - Primitive types are ADTs, user-defined types are objects
 - Rules of visibility
 - Private, package, protected, public
 - Objects of the same class can see inside each other (ADT property)
- Polymorphism in Java
 - **Static polymorphism**: Methods in the same class with the same name but different argument types (a.k.a. method overloading)
 - **Dynamic polymorphism**: Methods with the same name in different classes
- Inheritance in Java
 - Support for the **substitution principle**: an argument of a given class type will accept objects of any subclass
 - Support for multiple inheritance using a new concept called **interface** (a specific form of a general data abstraction interface)

Functional programming in Java



- Not much support for functional paradigm
 - More support is being added as Java evolves (lambda expressions in Java 8, which are procedure values)
 - Problem of legacy code!
 - Scala has full support for functional paradigm
- **Final attributes and variables**: can only be assigned once
 - Objects can be immutable, but are not functional objects
- **Final classes**: cannot be extended with inheritance
- **“inner classes”**: a class defined inside another class
 - An instance of an inner class is almost (but not completely) a procedure value