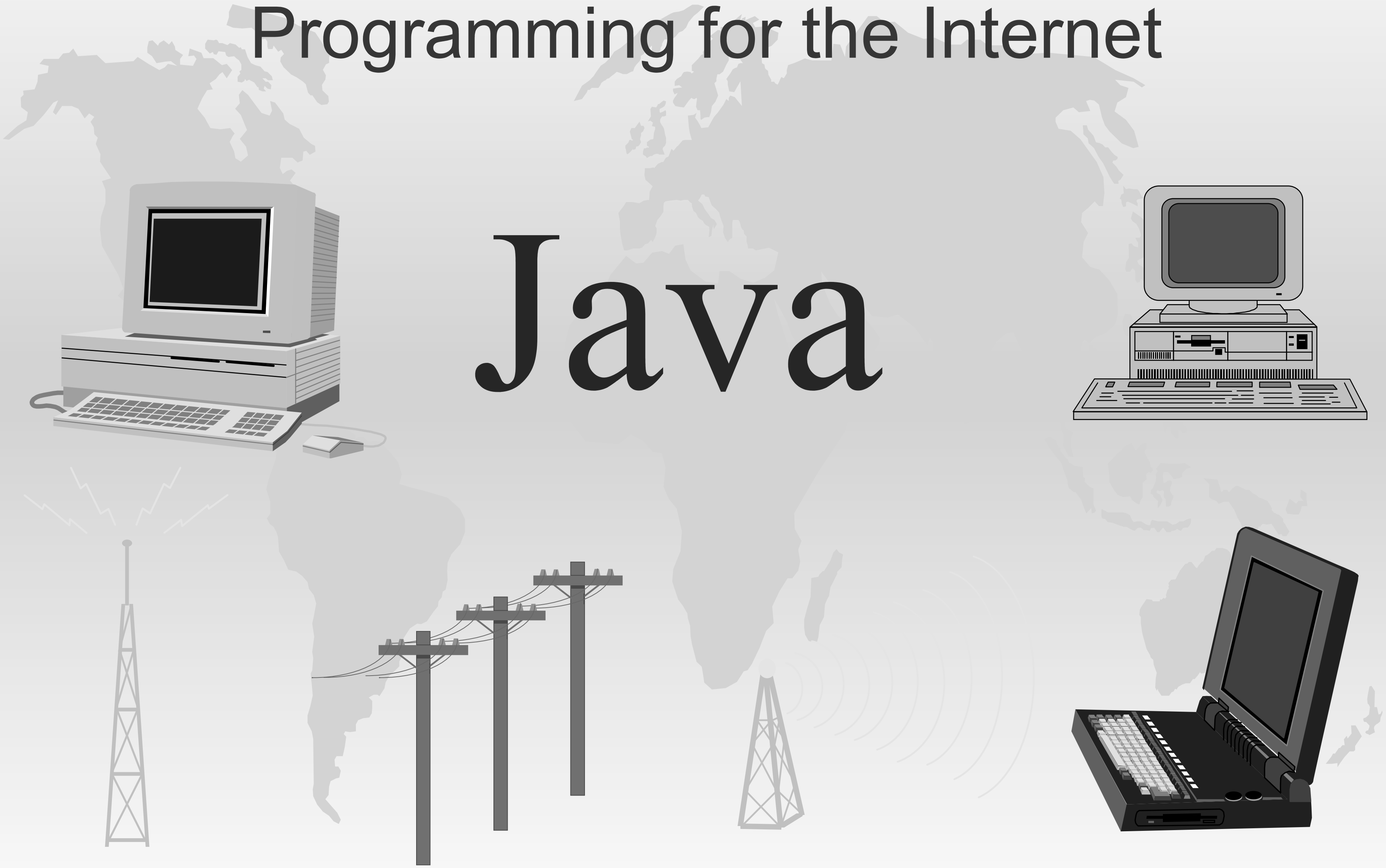
Programming for the Internet

Java



Design goals of Java

- Machine independence
- Platform independence
- Full specification
  - Implementation independence
- Object orientedness
- Simplicity

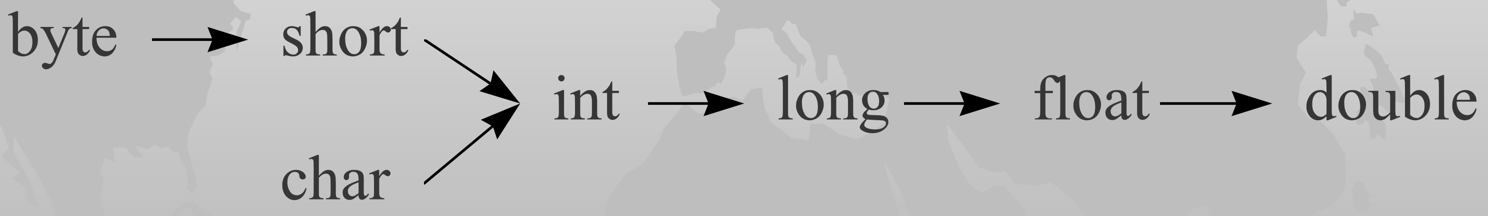
Types

- Primitve Types
  - – arithmetic
    - integral
    - floating-point
  - – character
  - – boolean
- Reference Types
  - – class type
  - – array type
  - – interface type

byte
short
int
long
float
double
char
boolean

Conversions of Predefined Types

widening

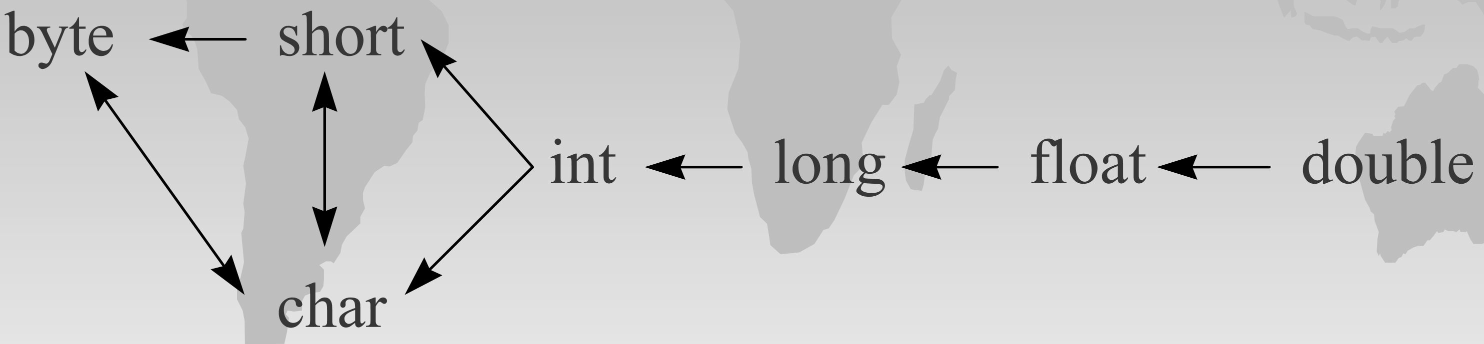

byte → short

short → int

char → int

int → long → float → double

narrowing


double → float → long → int

short → byte

int → short

int → char

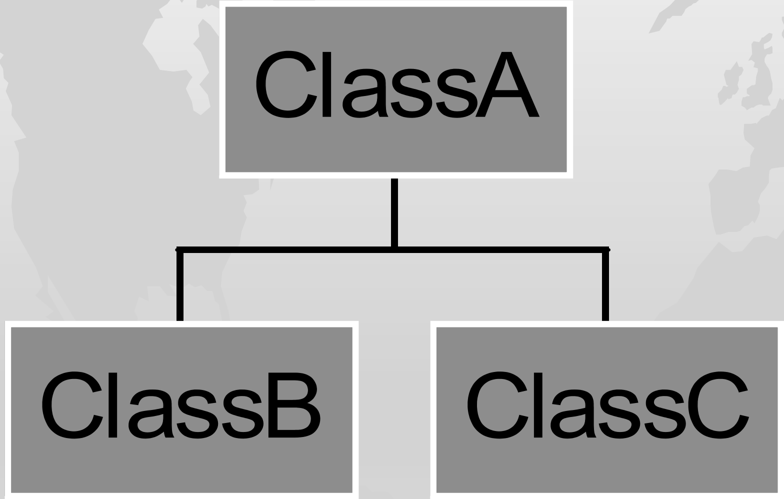
byte ↔ char

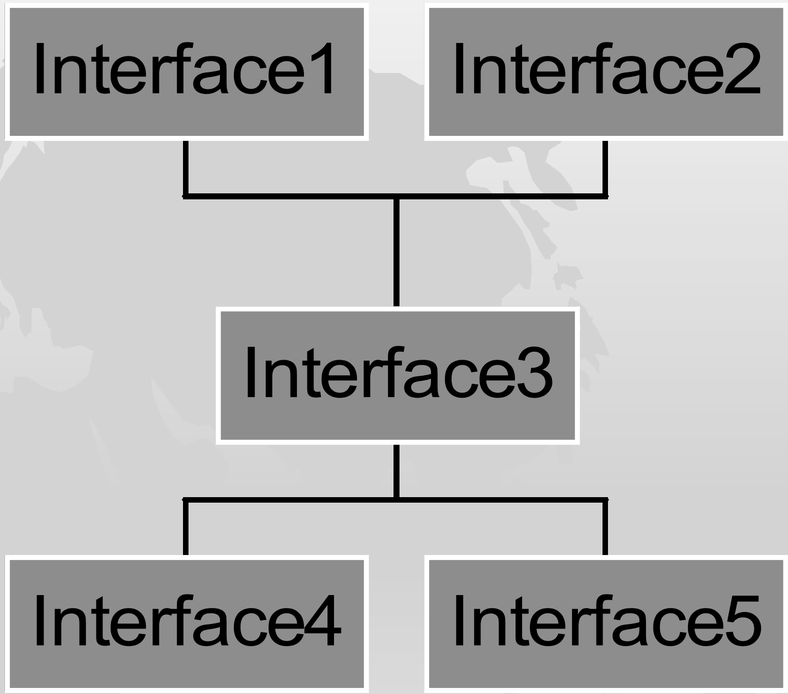
short ↔ char

Interface Declarations

- specifies a set of method signatures
- specifies associated named constants
- does not specify an implementation
- can be implemented by many classes
  - common interface
- implicitly abstract
- can be public

Classes and Interfaces



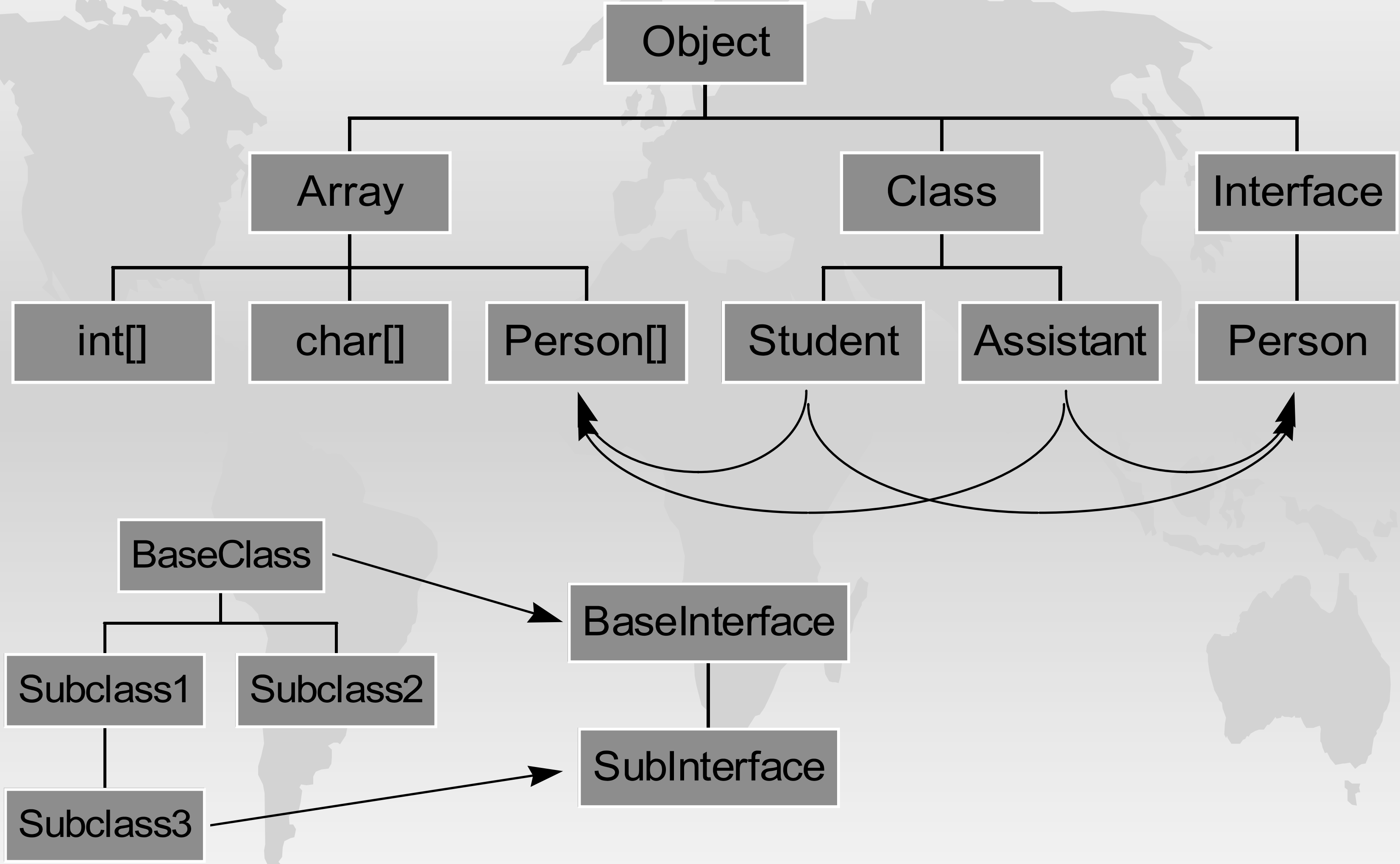


- single implementation inheritance
- code reuse

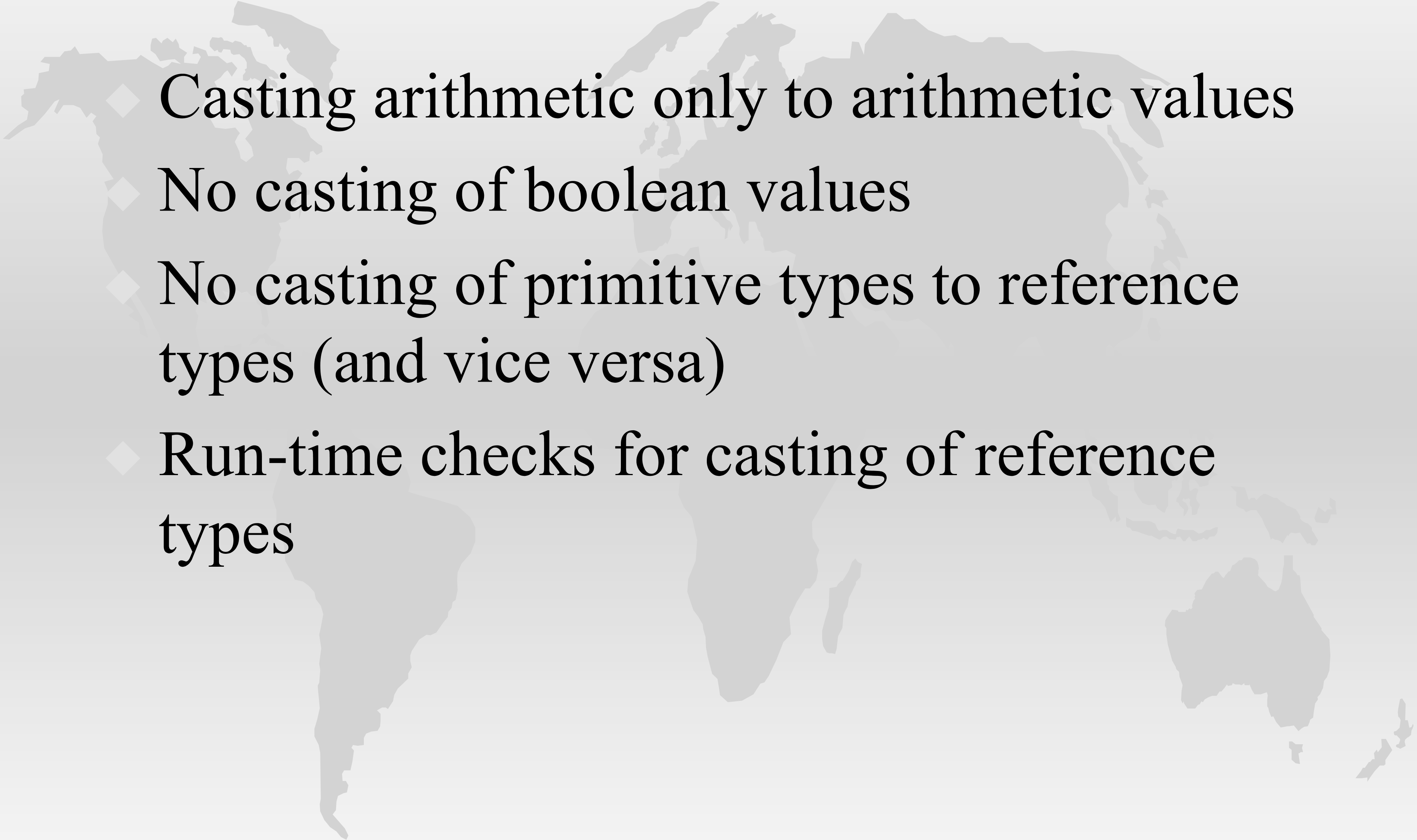
- (multiple) interface inheritance

class ClassA implements Interface3 {}
class ClassB extends ClassA implements Inferface4 {}

Conversions on Reference Values



Conversion - Casting



- Casting arithmetic only to arithmetic values
- No casting of boolean values
- No casting of primitive types to reference types (and vice versa)
- Run-time checks for casting of reference types

Name Spaces

- PackageName = {Ident "."} Ident.
  - correspond to directory structure
  - globally unique package names
    - COM.Sun.sunsoft.DOE, COM.Apple.quicktime.v2
- CompilationUnit = [PackageStmt] {ImportStmt} {TypeDecl}.
- TypeDecl = ClassDecl | InterfaceDecl.
- Parameter name space of methods
- Name space of local block
- Name space of nested local block

Access restrictions

- Access to package restricted by host system
- public - anywhere
- protected - in the package and in subclasses
- private - in the class only
- nothing - in package only
- Access to field names
  - access to package, public type
    - public field declaration
    - protected field declaration, use within subclass

Importing Packages and Types

- Packages
  - import *packageName*
  - import java.io
    - io.*name* or java.io.*name*
  - predefined package java.lang
    - Int, Float, Object, String, NullPointerException,...
- Types
  - import *packageName* . *Ident* ;
  - import java.util.Vector;
    - Vector or java.util.Vector
  - import *packageName* . * ;

Classes

- ClassDecl = {"abstract" | "final" | "public"} "class" Ident ["extends" className] ["implements" {interfaceName "," } interfaceName] ClassBody.
- ClassBody = "{" {FieldDecl} "}".
- FieldDecl = FieldVariableDecl | MethodDecl | ConstructorDecl | StaticInitializer.

Variable Declarations

- VariableDecl = {VariableModifier} typeName {VariableDeclarator}.
- VariableModifier = "public" | "protected" | "private" | "static" | "final" | "transient" | "volatile".
- VariableDeclarator = Ident {"[ ]"} "=" VariableInitializer.
- VariableInitializer = Expr | "{" {Expr ","} "}".

Method Declarations

- MethodDecl = [MethodModifier] resultType MethodDeclarator ["throws" {Exception ","} Exception] MethodBody.
- MethodModifier = "public" | "protected" | "private" | "static" | "abstract" | "final" | "native" | "synchronized".

MethodDeclarator = Ident "(" {TypeName Ident {"[ ]"} ","}.

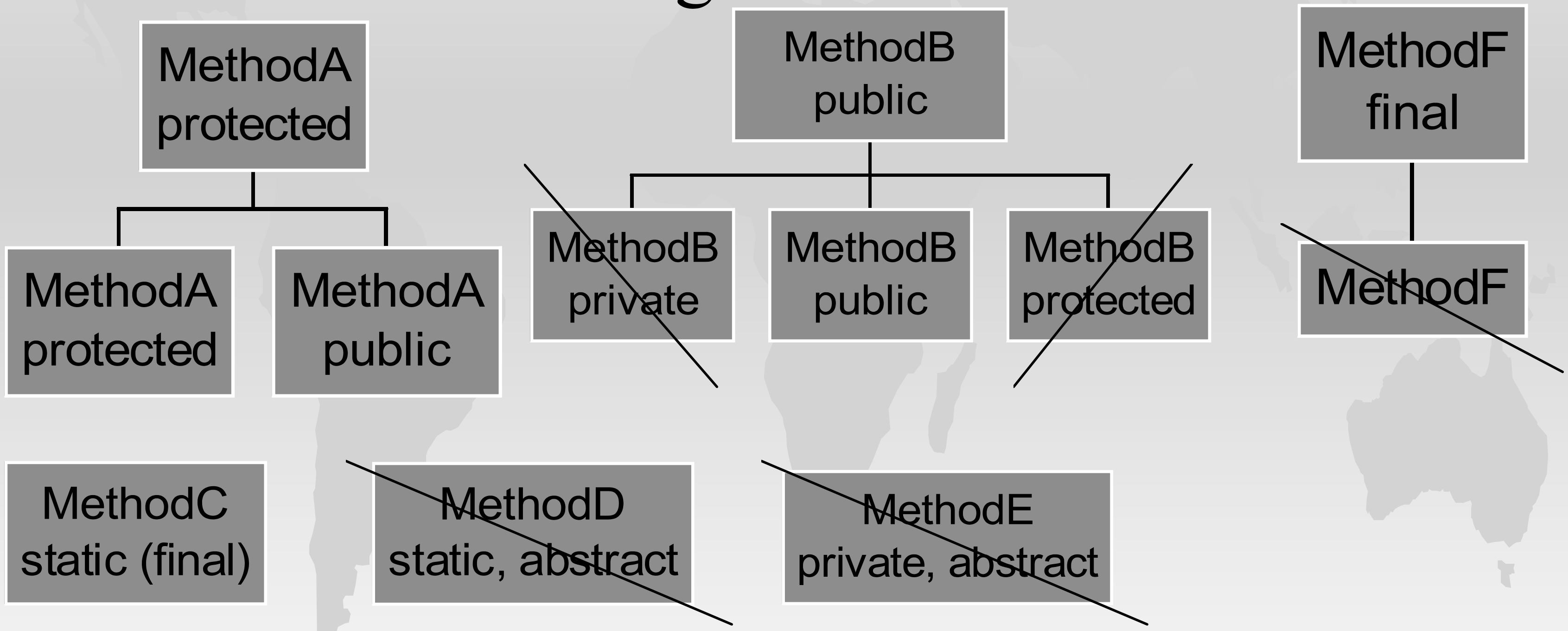
- MethodBody = Block | ";".

Method Modifiers

- public, protected, and private, or nothing
- abstract - no implementation, interface only
- static - class method
- final - not overridable
- synchronized - monitored execution
  - per object or per class (when static)

Method Overloading and Overriding

- Polymorphic method declarations
  - same name, different parameters, different parameter types
- Method overriding



MethodA protected

MethodA protected

MethodA public

MethodB public

~~MethodB private~~

MethodB public

~~MethodB protected~~

MethodF final

~~MethodF~~

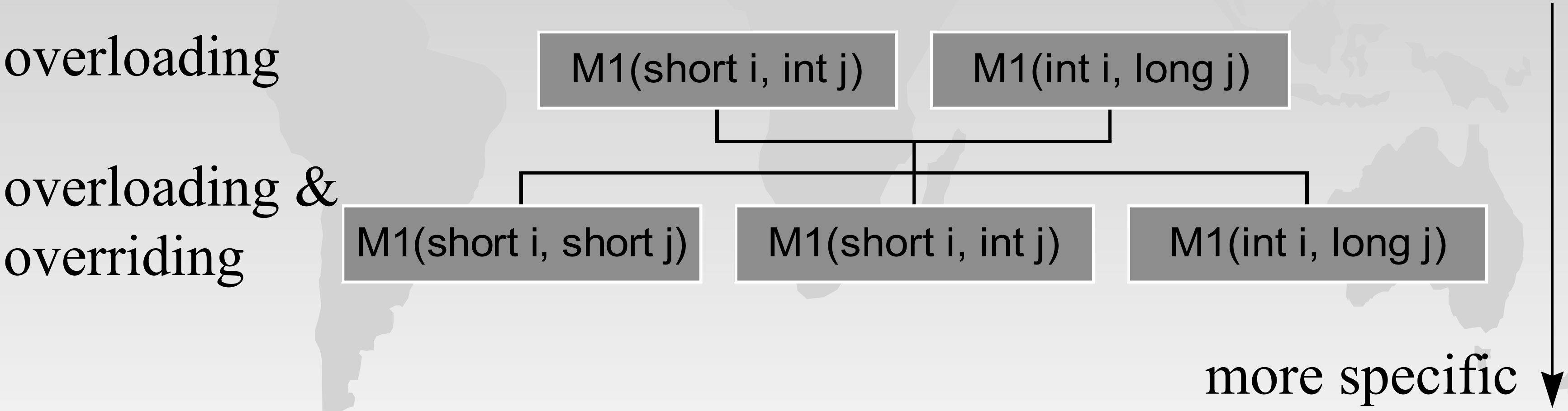
MethodC static (final)

~~MethodD static, abstract~~

~~MethodE private, abstract~~

Method Calls

- Method must be applicable
  - matching names
  - matching parameters (number & types)
- Method must be most specific



overloading

M1(short i, int j)

M1(int i, long j)

overloading &
overriding

M1(short i, short j)

M1(short i, int j)

M1(int i, long j)

more specific

Memory Management

- Object Creation
  - Student stud = new Student();
  - Person pers;
  - pers = stud;
  - int arr[] = {1, 4, 10, -4};
  - int arr[] = new int[100];
- Automatic storage reclamation (garbage collection)
- Finalization
  - non-static method *void finalize();*

Class Loading

- loaded when needed
  - implicitly
  - explicitly (*classLoader, Class.forName*)
- allocation of storage for static variables
- allocation of a class object (instance of class *Class*) to represent the class

Differences to Oberon-2 or C

- classes and interfaces only, no "normal" functions
- method dispatch not strictly dynamic
- change of dynamic type via casts
- no nested comments
- dangling else (associated with last elseless if)
- Exception handling
- name spaces and dynamic loading of classes
- one public type or interface per package
- Unicode character set for specification of programs

Programming for the Internet

Java