# Miscellaneous

- PA5 deadline extended
  - Dec. 30, 10:00PM
- Announcement on P/F
- Course evaluation 课程评估

# CS109 Introduction to Computer Programming

Instructor: Zhuozhao Li

lizz@sustech.edu.cn

# Disclaimer

- Course contents not mentioned in the slides may still be tested in the final exam
- Review the important concepts taught in this course at a high level. But you must understand the knowledge behind each concept in detail
- Scope: Everything except GUI programming and Exception

# Final Exam

- True or False
- Multiple choice
- Fill in the blank
- Output of Java
- Programming

# Computers and Programs

# Computer Organization

- Input unit
- Output unit
- Memory unit
- Central processing unit (CPU)
  - Control unit + Arithmetic/logic unit (ALU)
- Secondary storage unit

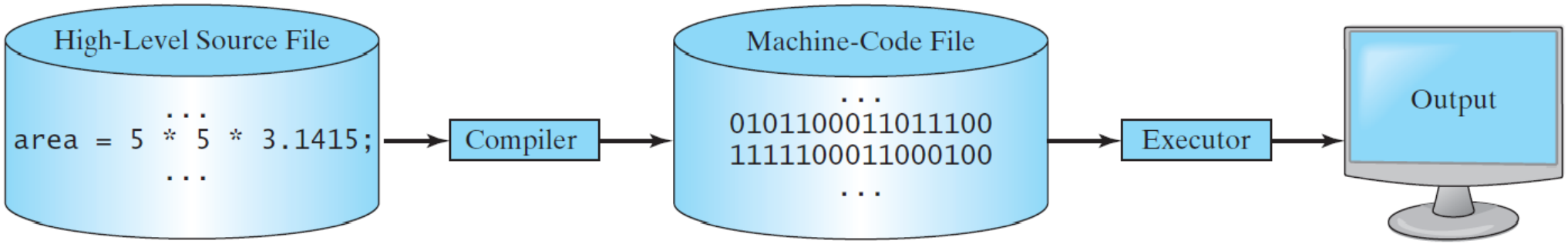
John von Neumann
(1903-1957)
Hungarian-American
mathematician, physicist

# Computer Program

- A computer program is a set of machine-readable instructions that tells a computer how to preform a specific task
- Programs are written in programming languages
- There are many programming languages
  - Low-level, understandable by a computer
  - High-level, needs a translator (compiler or interpreter)!

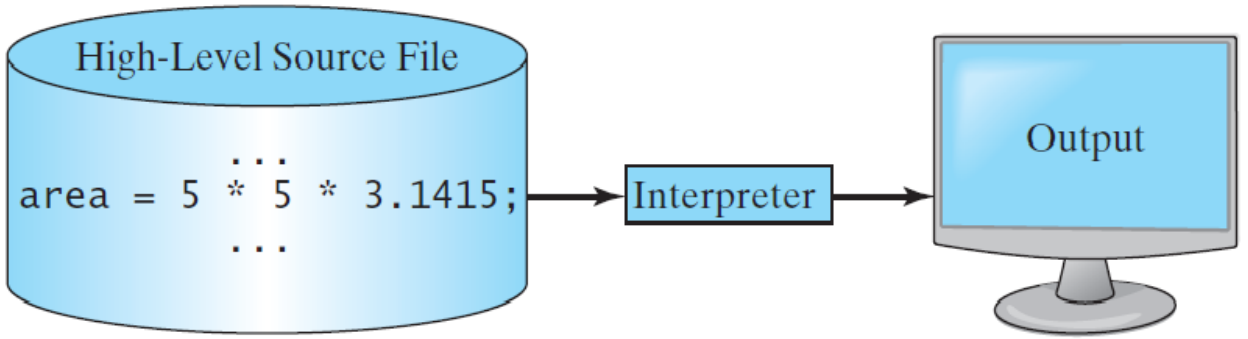
# Compilation: from source to executables

- A complier translates source programs into machine codes that can run directly on the target computer.



# Interpreter

- An interpreter directly translates and executes the statements in source code, without requiring the programs to be compiled into machine codes.



# Java Programs

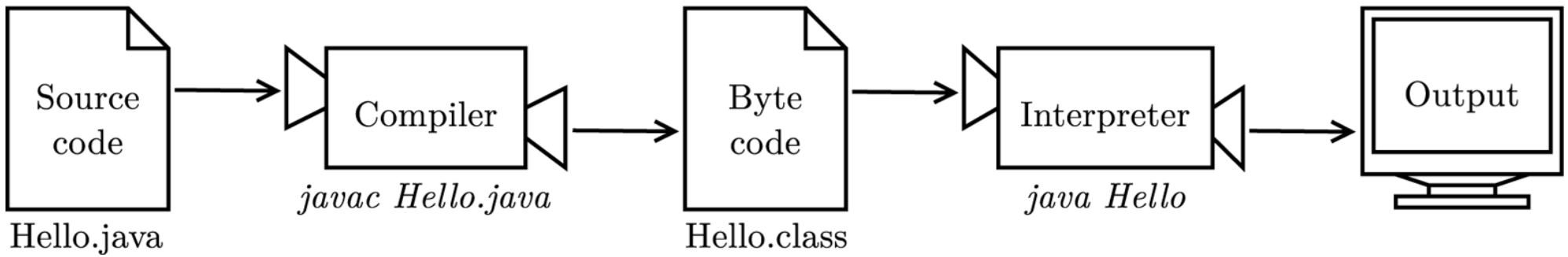
# Java

- Java is a general-purpose computer-programming language
- Java was originally developed by **James Gosling** at Sun Microsystems (acquired by Oracle) and released in 1995

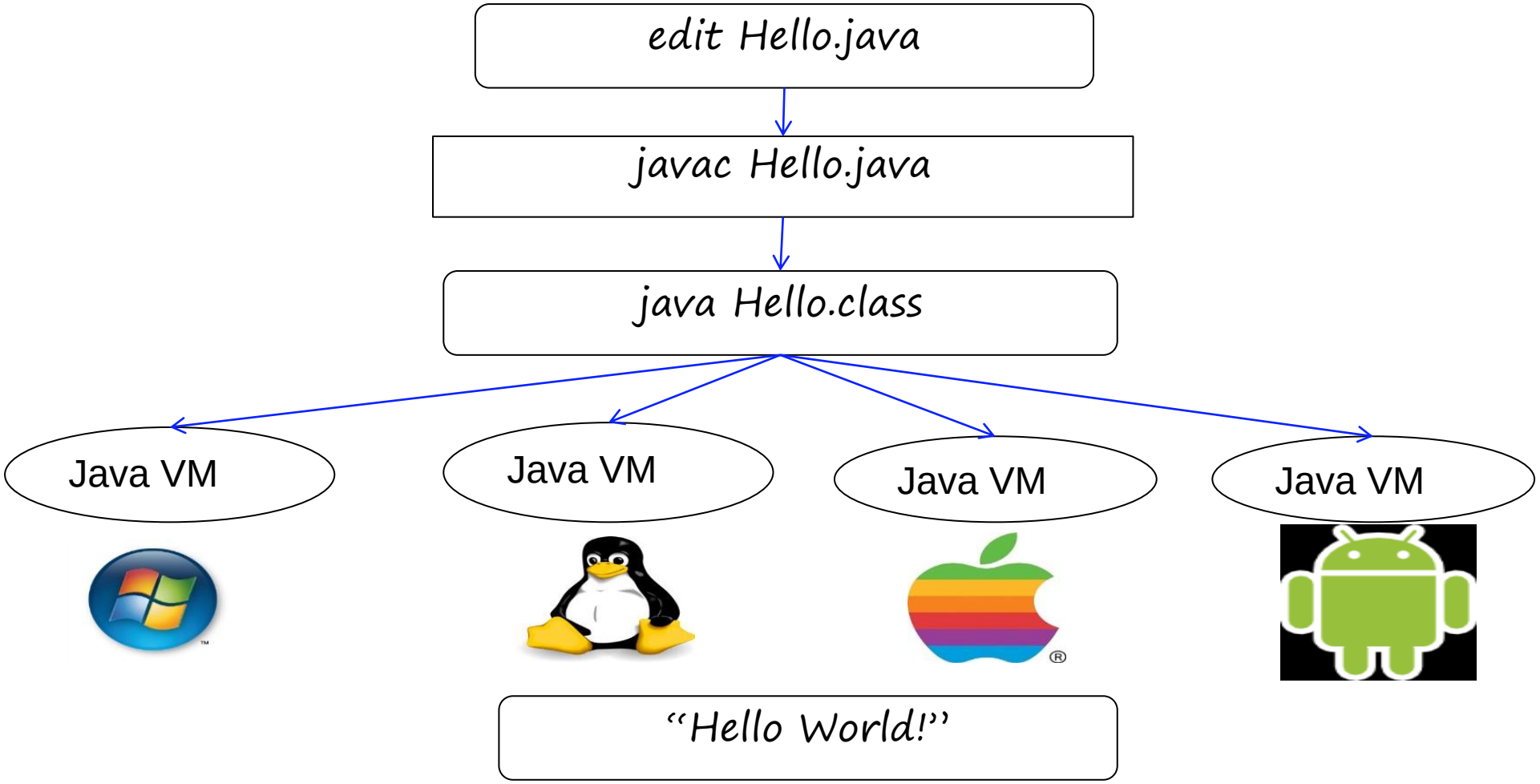
# JDK, JRE, and JVM

- **Java Development Kit (JDK)** is a software development environment for developing Java programs. It includes a Java Runtime Environment (JRE), an interpreter (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools.
- **Java Runtime Environment (JRE)** provides the minimum requirements for executing a Java program; it consists of a Java Virtual Machine (JVM), core classes, and supporting files.
- **Java Virtual Machine (JVM)** is an abstract computing machine that enables a computer to run a Java program.
- JDK = JRE + Development tools, JRE = JVM + Library classes

# Java is both compiled and interpreted



# Java is platform-independent



# Java Language Basics

# Identifiers 标识符

- A name in a Java program is called an **identifier**, which is used for identification purpose. It can be a class name, a method name, a variable name, or a label name
- The only allowed characters in Java identifiers are **a to z, A to Z, 0 to 9, $** and _(Underscore).
- Identifiers can't start with digit, e.g., **123name** is not valid.
- Java identifiers are case sensitive, e.g.,, **name, Name, NAME** are different

# Identifiers cont.

- We can't use reserved keywords as identifiers, i.e., int if = 20 will not compile

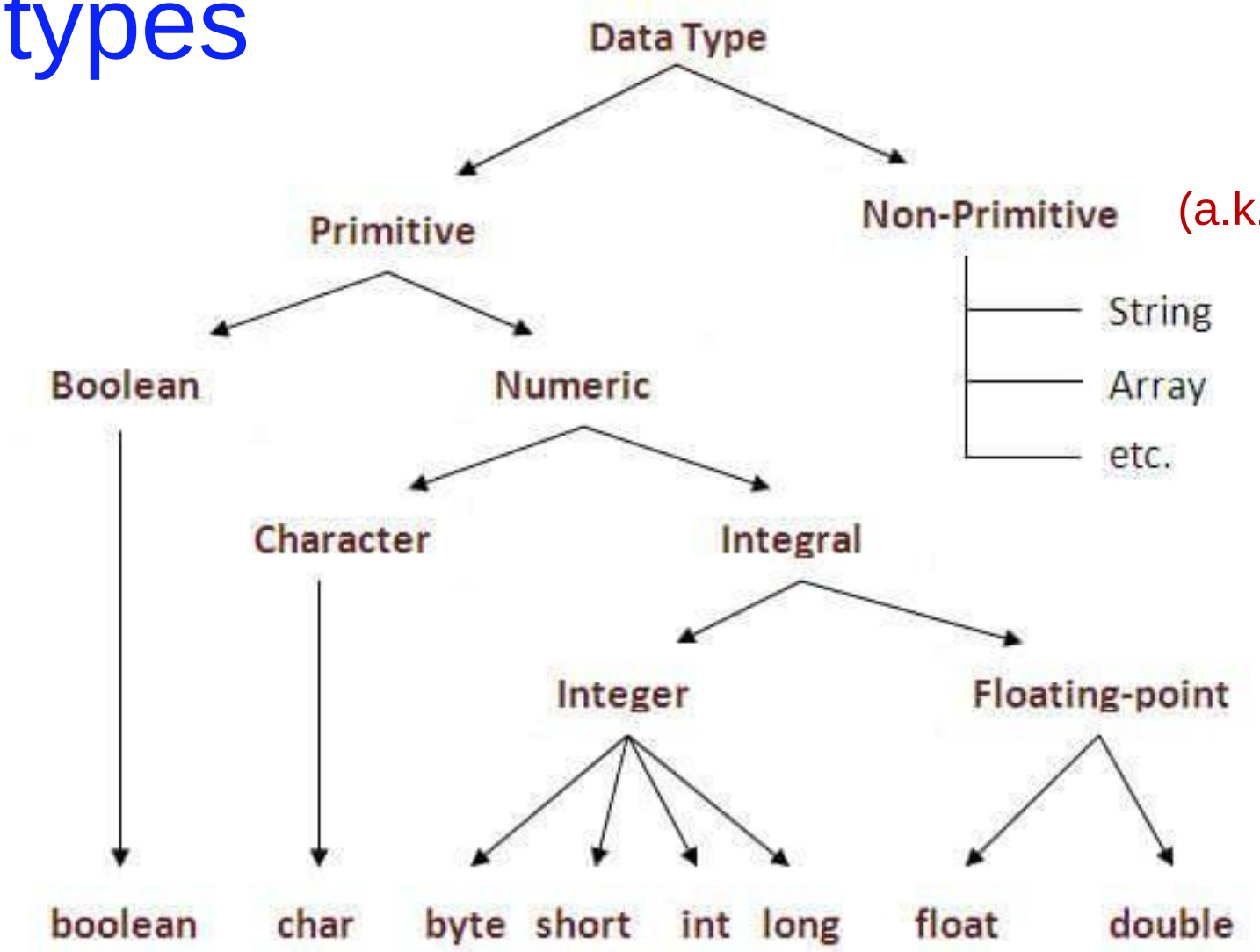
| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | switch |
| assert | default | goto | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp | volatile |
| const | float | native | super | while |

Keyword names are all in lower case

# Variable

- A variable is a container which holds data while a Java program executes
- It is the name of the memory location allocated for storing the data
- There are three kinds of variables in Java
  - Local variables: A variable declared inside a method (be careful with the scope)
  - Instance variable: A variable declared inside a class but outside the methods and not declared static
  - Static variable: A variable which is declared as static

# Data types




(a.k.a. reference types)

https://www.javatpoint.com/variable-datatype

# Type Cast （类型转换）

```
int total;
int studentCounter;
double average;
```

```
average = (double) total / studentCounter;
```

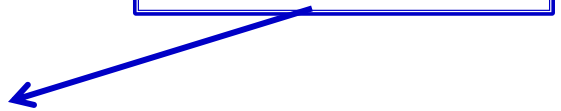
The unary cast operator creates a temporary floating-point copy of its operand

- Cast operator performs explicit conversion (or type cast).
- This precedence is one level higher than the binary arithmetic operator, e.g., *, / and %.
- The value stored in the operand is unchanged (e.g., the value of total is unchanged!!!)

# Type Promotion (类型提升)

```
int total;
int studentCounter;
double average;

average = (double) total / studentCounter;
```



Type promotion from int to double

- Java evaluates only arithmetic expressions in which the operands' types are identical.
- Promotion (or implicit 隐含 conversion) performed on operands.
- In the above expression, the `int` value of studentCounter is promoted to a `double` value for computation.

- byte->short->int->long->float->double

# More on reference types

- All non-primitive types are reference types, including instantiable classes and arrays
  - `Scanner, String, String[], int[]`
- Programs use reference-type variables to store the locations of objects in memory.
  - `GradeBook myGradeBook = new GradeBook(“CS102A”);`
  - Such a variable is said to refer to an object in the program.
- Reference-type variables, if not explicitly initialized, get the default value `null`

# Common arithmetic operators

| Operator | Description |
|---|---|
| + | Additive operator (also used for String concatenation) |
| - | Subtraction operator |
| * | Multiplication operator |
| / | Division operator |
| % | Remainder operator |
| ++ | Increment operator; increments a value by 1 |
| -- | Decrement operator; decrements a value by 1 |

- Integer division yields an integer quotient. The fractional part is simply discarded (3 / 2 = 1)
- int a = 6; int b = ++a; int c = a++; (a will be 8, b and c will be 7 after execution)

# Equality and relational operators

| Operator | Description |
|---|---|
| == | Equal to (do not confuse with the assignment operator =) |
| != | Not equal to |
| > | Greater than |
| >= | Greater than or equal |
| < | Less than |
| <= | Less than or equal |

# Common logical operators

- **Logical operators** help form complex conditions by combining simple ones:
  - && (conditional AND, **short-circuit behavior**)
  - || (conditional OR, **short-circuit behavior**)
  - & (boolean logical AND)
  - | (boolean logical inclusive OR)
  - ^ (boolean logical exclusive OR)
  - ! (logical NOT)

# The Operators Introduced So Far

Precedence

| Operators | Associativity | Type |
|---|---|---|
| `++ --` | right to left | unary postfix |
| `++ -- + - ! (type)` | right to left | unary prefix |
| `* / %` | left to right | multiplicative |
| `+ -` | left to right | additive |
| `< <= > >=` | left to right | relational |
| `== !=` | left to right | equality |
| `&` | left to right | boolean logical AND |
| `^` | left to right | boolean logical exclusive OR |
| `\|` | left to right | boolean logical inclusive OR |
| `&&` | left to right | conditional AND |
| `\|\|` | left to right | conditional OR |
| `?:` | right to left | conditional |
| `= += -= *= /= %=` | right to left | assignment |

**Fig. 4.18** | Precedence/associativity of the operators discussed so far.

# Structured Programming

- Only three forms of control are needed to implement any algorithm:
  - Sequence
  - Selection
  - Repetition

# Structured Programming

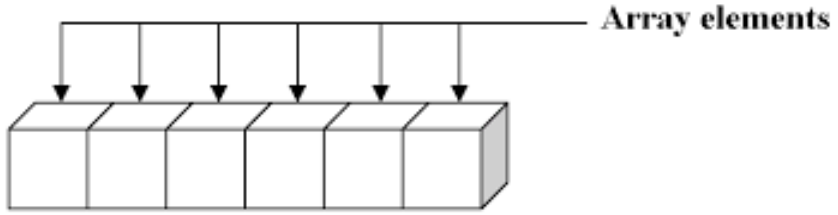
- Selection is implemented in one of three ways:
  - `if` statement (single selection)
  - `if…else` statement (double selection)
  - `switch` statement (multiple selection)
- The simple `if` statement is sufficient to provide any form of selection. Everything that can be done with the `if…else` and `switch` can be implemented by combining `if` statements.

```
String result = studentGrade >= 60 ? "Passed" : "Failed"
```

# Structured Programming

- Repetition is implemented in one of three ways:
  - `while` statement
  - `do…while` statement
  - `for` statement
- The `while` statement is sufficient to provide any form of repetition. Everything that can be done with `do…while` and `for` can be done with the `while` statement.
- Differences between `break` and `continue` statements.
- Enhanced `for` statement, `for (int a : arrays) System.out.println(a);`

# Arrays




- An **array** is a group of variables (elements) containing values of the same type.
- Arrays are objects, so they're considered reference types. Arrays are created by the keyword `new`
- Elements can be either primitive types or reference types. They can be retrieved using indexes
- Valid array indexes: 0 to array.length - 1 (runtime exception will occur when using other values)
- Unlike collections such as ArrayList, the capacity of an array is fixed once created

# Strings

- A string is a sequence of characters
- A string is an object of the class `java.lang.String`
- Strings can be created by using string literals or various String constructors
  - `String s = “hello world”; String s1= new String(“hello world”);`
- String objects are immutable. Any modification creates a new String object.
- String class provides many useful methods: length(), charAt(), substring()
- The equals() method tests whether two strings are identical (NOT ==, which only works for primitive types)

# Enum types

- Enum is a special data type that enables a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it.
- All enum types are reference types
- Enum constants are implicitly `final` and `static`
- Any attempt to create an object of an enum type with operator `new` results in a compilation error (enum constructor should be private or package-private)
- Besides enum constants, enums can also contain members such as constructors, fields and methods

# Classes and Objects

# Classes and Objects

- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

- **Object** − An object is an instance of a class. Objects have states and behaviors.
  - Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating.

# Classes

- A class can contain member variables (instance and static ones) and methods (instance and static ones)

- Differences between instance variables/methods and static variables/methods

# Access Level Modifiers

| Modifier | Class | Package | Subclass | World |
|---|---|---|---|---|
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| no modifier | Y | Y | N | N |
| private | Y | N | N | N |

Note that this is for controlling access to class members. At the top level, a class can only be declared as `public` or package-private (no explicit modifier)

# Methods

- Method parameters are local variables (only visible in the method)
- **Parameter vs. arguments:** A parameter is the variable which is part of the method's declaration. An argument is an expression used when calling the method.
- When calling methods, all arguments are **passed by value**. A method call can pass two types of values to the called method: copies of primitive values and copies of references to objects.

# Method overloading

- Methods of the same name can be declared in the same class, as long as they have different sets of parameters
- Used to create several methods that perform the same/similar tasks on different types or different numbers of arguments
- Compiler distinguishes overloaded methods by their **signature**: a combination of the method's name and the number, types and order of its parameters (return type is not part of method signature)

# Constructors

- A constructor is a special method used to initialize the data of an object
- Java compiler will create a default no-argument constructor for a class that has no explicitly-defined constructors
- The compiler will not create a default no-argument constructor for a class that has any explicitly-defined constructor
- Constructors can be overloaded to allow different ways of object creation

# The `this` keyword

- The main use of `this` reference is to differentiate the formal parameters of methods and the fields of classes (when the fields are shadowed by the method parameters)

- "`this`" can also be used to invoke another constructor of the same class in the body of one constructor

# The `static` keyword

- Recall that every object of a class has its own copy of all the instance variables of the class.
  - Instance variables represent concepts that are unique per instance, e.g., `name` in class `Student`.

- In certain cases, only one copy of a particular variable should be shared by all objects of a class (e.g., a counter that keeps track of every object created for memory management).
  - A `static` field—called a class variable—is used in such cases.

- A `static` method cannot access non-static class members (e.g., instance variables), because a static method can be called even when no objects of the class have been instantiated.

# Inheritance and polymorphism

# Inheritance

- Java only allows single inheritance, but one class can implement several interfaces

```
public class SubClass extends SuperClass implements Interface1, Interface2 { … }
```

- A subclass inherits all of the public and protected members of its parent, no matter what package the subclass is in
- If the subclass is in the same package as its parent, it also inherits the package-private members of the parent
- Private members of the superclass cannot be inherited, constructors are not class members and cannot be inherited
- Every class directly or indirectly extends `java.lang.Object`

# Method Overriding

- A subclass can override a method it inherits from its parent to provide more specific implementation (final methods cannot be overridden, private methods and static methods are implicitly final)
- An overridden method must have the same signature as the superclass method
- An overridden method may have a more specific return type
- The access level of an overriding method can be higher, but not lower than that of the overridden method (package-private < `protected` < `public`)

# The `super` keyword

- The main use of the `super` keyword is to invoke a superclass constructor in a subclass constructor

- "`super`" can also be used to invoke overridden methods of the superclass (when the subclass overrides a method of its superclass)

# Subclass Constructors

- Each constructor in the subclass needs to invoke a superclass constructor for object construction (e.g., to initialize inherited instance variables) by using the super keyword.
- If this is not explicitly done, the compiler automatically inserts a call to the no-argument constructor of the superclass. If the super class does not have a no-argument constructor, you will get a compile-time error.
- **Constructor chaining:** If a subclass constructor invokes a constructor of its superclass, there will be a whole chain of constructors called, all the way back to the constructor of Object. You need to be aware of its effect.

# Polymorphism

- In Java, polymorphism is the ability of an object to take different forms. All java objects are polymorphic and can pass multiple IS-A tests

- With polymorphism, an object of a subclass can be treated as an object of the superclass

- Objects of different types can be accessed through the same interface. Each type can provide its own implementation of this interface

# Assignment between superclass and subclass variables

- Assigning a superclass object's reference to a superclass variable is natural
- Assigning subclass object's reference to a subclass variable is natural
- Assigning a subclass object's reference to a superclass variable is safe (the superclass variable can be used to access only superclass members)
- Assigning a superclass object's reference to a subclass variable leads to compilation errors

# Dynamic binding (or late binding)

- When Java compiler encounters a method call made through a reference variable, it determines if the method can be called by checking the variable's class type
  - If that class contains the proper method declaration (or inherits one), the call will be successfully compiled
- At execution time, the type of the object to which the variable refers determines the actual method to use (dynamic binding)

# Static binding (or early binding)

- In Java, final methods in a super class cannot be overridden in the subclass. Private methods and static methods are implicitly final.

- A final method's declaration cannot change so the calls to the final methods are resolved statically at compile time (static binding)

# Abstract class

- A class that represents a generalization and provides functionality but is only intended to be extended and not instantiated
- An abstract class can contain instance/static variables/methods and constructors
- Usually contain one or more abstract methods that are intended to be overridden by subclasses
- Subclass can become concrete only if it implements all inherited abstract methods; otherwise, it has to be declared as abstract

# Abstract class

- Although abstract classes cannot be used to instantiate objects, they can be used to declare variables. Abstract superclass variables can hold references to objects of any concrete classes derived from them.

- Note that we can use abstract class names to invoke static methods declared in those abstract classes (since invoking static methods do not require the existence of objects)

# Interface

- An interface describes a set of methods that can be called on an object, but does not provide concrete implementations for all the methods

- An interface is often used when unrelated classes need to share common methods and constants (interfaces do not enforce a class relationship)

- An interface is a reference type. When a class implements an interface, it has an IS-A relationship with the interface data type

# Interface declaration

- An interface contains only constants and abstract methods (Java 7 or earlier)
  - All fields are implicitly public static final
  - All methods are implicitly public abstract
- An interface can extend multiple interfaces
- An interface cannot have a constructor
- An interface can be used to define variables. In such cases, any object you assign to the variable must be an instance of a class that implements the interface

# Generic class/method and collections

# Generics

- Generic class and methods in Java parameterize data types that can be used in the methods or class
- Type parameters can represent only reference types (not primitive types)
- They help make your program more type safe (compilers does type inference and inserts safe cast operations when necessary)
- Type Erasure

# Java Collections Framework (JCF)

- JCF is a set of **classes** and **interfaces** that implement reusable **data structures** to help **group and manage related objects**
- Similar to arrays, collections hold references to objects that can be managed as a group (**one object represents a group of objects**)
- Unlike arrays, collections do not need to be assigned a certain capacity when instantiated. **Their size can grow and shrink automatically** when objects are added or removed.
- Unlike arrays, **collections cannot hold primitive type elements** (e.g., `int`), they can only hold object references (arrays can do both).

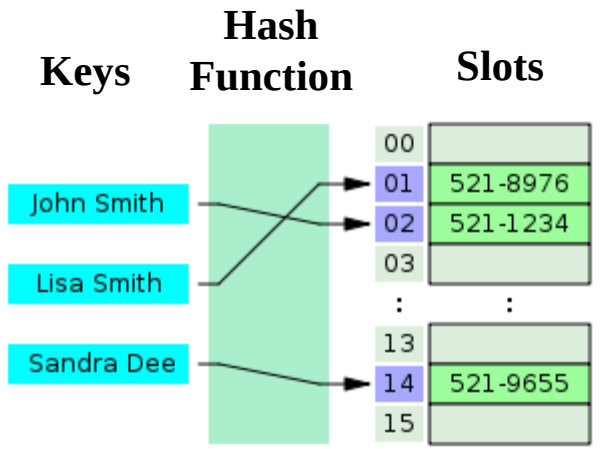
https://en.wikipedia.org/wiki/Java_collections_framework

# `List` and `Set`

- `Collection` has two important offspring: `List` and `Set`
- **A list is an ordered collection** (also known as a **sequence**). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Lists typically allow duplicate elements.
- **A set is collection that contains no duplicate elements**. This interface models the mathematical set abstraction

# HashMap （哈希表）

- `HashMap` (hash table) is a data structure that can map keys to values. It is a concrete implementation of the `Map` interface.
- It uses a **hash function （哈希函数）** to compute an index into an array of slots, from which the desired value can be found (**very efficient**).
- A **hash function** is any function that can map data of arbitrary size to data of fixed size. Well-defined hash functions have low chances of collisions (mapping two different keys to the same hash values).



# Thank you!

Please submit the course evaluation