# Chapter 1 Java Building Blocks

## Java Class Structure

### Fields and Methods

### Comments

### Classes vs. Files

## main() Method

## Package Declarations and Imports

### Wildcards

### Redundant Imports

### Naming Conflicts

### Creating New Package

## Creating Objects

### Constructors

### Reading/Writing Object Fields

### Instance Initializer Blocks

### Order of Initialization

## Object References vs. Primitive

### Primitive Types

### Reference Types

### Key Differences

## Declaring and Initializing Variables

### Declaring Multiple Variables

### Identifiers

## Default Initialization of Variables

### Local Variables

### Instance and Class Variables

## Variable Scope

## Ordering Elements in a Class

## Destroying Objects

### Garbage Collection

### finalize()

## Benefits of Java

# Chapter 2 Operators and Statements

## Java Operators

## Binary Arithmetic Operators

### Arithmetic Operators

### Numeric Promotion

## Unary Operators

### Logical Complement and Negation Operators

### Increment/Decrement Operators

## Additional Binary Operators

### Assignment Operators

### Compound Assignment Operators

### Relational Operators

### Logical Operators

### Equality Operators

## Java Statements

### if-then

### if-then-else

### switch

### while

### do-while

### for

## Advanced Flow Control

### Nested Loops

### Optional Labels

### break Statement

### continue Statement

# Chapter 3 Core Java APIs

## Creating and Manipulating Strings

### Concatenation

### Immutability

### String Pool

### Important String Methods

### Method Chaining

## StringBuilder Class

### Mutability and Chaining

### Creating StringBuilder

### Important StringBuilder Methods

### StringBuilder vs. StringBuffer

## Equality

## Java Array

### Array of Primitive

### Array with Reference Variables

### Using Array

### Sorting

### Searching

### Varargs

### Multidimensional Arrays

## ArrayList

### Creating ArrayList

### Using ArrayList

### Wrapper Classes

### Autoboxing

### Coverting Between array and List

### Sorting

## Dates and Times

### Creating Dates and Times

### Manipulating Dates and Times

### Working with Periods

### Formatting Dates and Times

### Parsing Dates and Times

# Chapter 4 Methods and Encapsulation

## Designing Methods

### Optional Specifiers

### Return Type

### Method Name

### Parameter List

### Optional Exception List

### Method Body

## Varargs

## Access Modifiers

### Private

### Default(Package Private)

### Protected

### Public

### Designing Static Methods and Fields

### Calling Static Variable or Method

### Static vs. Instance

### Static Variables

### Static Initialization

### Static Imports

## Passing Data Among Methods

## Overloading Methods

## Constructors

### Default Constructors

### Overloading Constructors

### Final Fields

### Order of Initialization

## Encapsulating Data

### Creating Immutable Classes

## Simple Lambdas

### Example

### Syntax

### Predicates

# Chapter 5 Class Design

## Class Inheritance

Support single inheritance.

Prevent a class from being extended using final modifier.

### Extending a Class

Only one public class per file.

public abstract class ElephantSeal extends Seal {

// Methods and Variables

}

### Applying Class Access Modifiers

public and default(package-level private) are only modifiers for top-level classes in Java file.

protected and private can only be applied to inner classes.

Java file can have at most one public class or interface. It can have no public class at all.

### Creating Java Objects

java.lang.Object is the only class that doesn’t have any parent classes.

All classes are inherited from java.lang.Object.

Defining Constructors

super() command may only be used as the first statement of the constructor.

Child class can use any valid parent class constructor.

**Compiler Enhancements**

Java compiler automatically inserts a call to the no-argument constructor super(), if the first statement is not a call to the parent constructor.

If parent class doesn’t have no-argument constructor, Java compiler will not help and you must create at least one constructor in child class that explicitly calls a parent constructor via the super() command.

Be wary if parent class defines a constructor that takes arguments and doesn’t define no-argument constructor.

**Constructor Definition Rules**

1. The first statement of every constructor is a call to another constructor within the class using this(), or a call to a constructor in the direct parent class using super().
2. The super() call may not be used after the first statement of the constructor.
3. If no super() call is declared in a constructor, Java will insert a no-argument super() as the first statement of the constructor.
4. If the parent doesn’t have a no-argument constructor and the child doesn’t define any constructors, the compiler will throw an error and try to insert a default no-argument constructor into the child class.
5. If the parent doesn’t have a no-argument constructor, the compiler requires an explicit call to a parent constructor in each child constructor.

### Calling Inherited Class Members

Parent constructor is always executed before the child constructor.

If the child class overrides a member of the parent class, this and super could have different effects when applied to a class member.

*super() vs. super*

super() is a statement to call parent constructor, and may only be used in the first line of a constructor of a child class.

super is a keyword used to reference a member defined in a parent class and may be used throughout the child class.

### Inheriting Methods

**Overriding a Method**

Compiler performs following checks when you override a nonprivate method:

1. The method in the child class must have the same signature as the method in the parent class.
2. The method in the child class must be at least accessible or more accessible than the method in the parent class.
3. The method in the child class may not throw a checked exception that is new or broader than the class of any exception thrown in the parent class method.
4. If the method returns a value, it must be the same or a subclass of the method in the parent class, known as *covariant return types*.

These rules are needed for consistency of the language in terms of polymorphism.

**Redeclaring Private Methods**

The redeclared version in child class is a separate and independent method, unrelated to the parent version’s method.

**Hiding Static Methods**

A hidden method occurs when a child class defines a static method with the same name and signature as a static method defined in a parent class.

1. The method in the child class must have the same signature as the method in the parent class.
2. The method in the child class must be at least accessible or more accessible than the method in the parent class.
3. The method in the child class may not throw a checked exception that is new or broader than the class of any exception thrown in the parent class method.
4. If the method returns a value, it must be the same or subclass of the method in the parent class, known as covariant return types.
5. The method defined in the child class must be marked as static if it is marked as static in the parent class (method hiding). Likewise, the method must not be marked as static in the child class if it is not marked as static in the parent class (method overriding).

Avoid hiding static methods in real codes. You should not reuse the name of a static method in your class if it is already used in the parent class

**Overriding vs. Hiding Methods**

At runtime the parent version of a hidden method is always executed if the call to the method is defined in the parent class.

**Creating Final Methods**

final method cannot be overridden.

The final modifier is only used on methods when the author of the parent method wants to guarantee very precise behavior.

### Inheriting Variables

Java doesn’t allow variables to be overridden but instead hidden.

**Hiding Variables**

The hidden variable has the same name with the parent class.

Child class has two copies of the variable and can access the parent version using super keyword.

*Don’t Hide Variables in Practice*

## Abstract Classes

An abstract class is a class that is marked with the abstract keyword and cannot be instantiated.

An abstract method is a method marked with the abstract keyword defined in an abstract class, for which no implementation is provided in the class in which it is declared.

### Defining Abstract Class

An abstract class may include non-abstract methods and variables. An abstract class is not required to include and abstract methods.

Abstract methods may only be defined in an abstract class. Abstract methods should not have {}. Abstract method should not be final or private.

### Creating Concrete Class

A concrete class is the first non-abstract subclass that extends an abstract class and is required to implement all inherited abstract method.

### Extending Abstract Class

## Interfaces

### Defining Interface

### Inheriting Interface

### Interface Variables

### Default Interface Methods

### Static Interface Methods

## Polymorphism

### Object vs. Reference

### Casting Objects

### Virtual Methods

### Polymorphic Parameters

### Polymorphism and Method Overriding

# Chapter 6 Exceptions

## Exceptions

### Role of Exceptions

### Understanding Exception Types

### Throwing an Exception

## try Statement

### finally Block

### Catching Various Types of Exceptions

### Throwing a Second Exception

## Common Exception Types

### Runtime Exceptions

### Checked Exceptions

### Errors

## Calling Methods That Throw Exceptions

### Subclasses

### Printing Exception