CS106 Data Structure & Java Fundamentals

BitTiger丨来自硅谷的终身学习平台

Claire & Bob

太阁BitTiger
BitTiger.io
BitTiger
BitTiger

BITTIGER

版权声明

所有太阁官方网站以及在第三方平台课程中所产生的课程内容，如文本，图形，徽标，按钮图标，图像，音频剪辑，视频剪辑，直播流，数字下载，数据编辑和软件均属于太阁所有并受版权法保护。

对于任何尝试散播或转售BitTiger的所属资料的行为，太阁将采取适当的法律行动。

我们非常感谢您尊重我们的版权内容。

有关详情，请参阅
https://www.bittiger.io/termsofuse
https://www.bittiger.io/termsofservice

BITTIGER

Copyright Policy

All content included on the Site or third-party platforms as part of the class, such as text, graphics, logos, button icons, images, audio clips, video clips, live streams, digital downloads, data compilations, and software, is the property of BitTiger or its content suppliers and protected by copyright laws.

Any attempt to redistribute or resell BitTiger content will result in the appropriate legal action being taken.

We thank you in advance for respecting our copyrighted content.For more info see https://www.bittiger.io/termsofuse and https://www.bittiger.io/termsofservice

Syllabus & Timeline

4.7	4.8	4.14	4.15
Java Fundamentals - Class - Static, final - **OOP** - **Multithreading** - **Generic**	**Data Structure** - Array - LinkedList - Tree - Stack/Heap	**Algorithm** - Binary Search - Sorting - BFS & DFS - DP	**Project & Java 8 & Java 9** - Project - Lambda - Functional Interface - Stream API

Q & A

- Java vs C++
- Intellij vs Eclipse
- Package

Package

- Similar to “Directory” in file systems
- Prevent name conflicts
- Control access
- Categorize classes

package package1.package2 → classpath/package1/package2

Object Oriented Programming

4 Fundamental Concepts

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Inheritance

- SubClass acquire properties from SuperClass
- Java does not support multi-inheritance -> can only **extend** from **one** class

Inheritance

```
public class Car extends Vehicle {
  private String model;

  public Car(String owner, String model, double speed) {
    super(owner, speed);
    this.model = model;
  }
}
```

```
public class Vehicle {
  private String owner;
  private double speed;

  public Vehicle(String owner, double speed) {
    this.owner = owner;
    this.speed = speed;
  }
}
```

Overriding

```
public class Vehicle {
    ...

    public void start() {
        System.out.println("Vehicle start...");
    }
}

public class Car extends Vehicle {
    ...

    @Override
    public void start() {
        System.out.println("Car start...");
    }
}
```

Overriding vs Overloading

- Overloading: allow a class to have multiple methods of the same name

```
public Vehicle() {
  owner = "";
  speed = 0.0;
}
```

```
public Vehicle(String owner, double speed) {
  this.owner = owner;
  this.speed = speed;
}
```

Object Class

- **Every** class has Object as a superclass
- All objects implement the methods of this class
  - finalize()
  - equals(Object obj)
  - hashCode()
  - toString()
  - ...

Object Class - Methods

- protected void finalize()
  - garbage collector determines there's no more reference to the object
- boolean equals(Object obj)
- int hashCode()

final vs finally vs finalize

Object Class - Methods

- protected void finalize()
- boolean equals(Object obj)
- int hashCode()

equals & hashCode

- Consistency -> called multiple times, same result
- If two objects equals() to each other, their hashCode() **must** return identical integer
- Necessary to override hashCode() once override equals()

equals & hashCode

```
/* Object.equals() */
public boolean equals(Object obj) {
  return (this == obj);
 }

/* Object.hashCode() */
public native int hashCode();
```

native implementation which provides the memory address to a certain extent

Polymorphism

- Single object -> multiple forms

Abstraction

- Provide a generalization over a set of behaviors
- User cares about high-level functionality not implementation
- abstract class & interface

Interface could have method implements?

Java 8 - Default Method

```
interface Vehicle {

    String getOwner();

    default void start() {
      System.out.println(
        String.format("%s is starting vehicle...", getOwner()));
    }
}
```

Interfaces vs Abstract Classes

- Interfaces
  - Do not maintain state (all variables are static final)
  - Multiple inheritance
- Abstract Class
  - Share code among classes
  - Share common non-public methods, variables
  - Share code which could change the state of the object

Interface ⇒ Description

Abstract Class ⇒ Code Sharing Basis

Encapsulation

- Hiding implementation details
- Easy to change implementation with new requirements
- Control “who” can access “what”

- *"Whatever changes encapsulate it"*

Getters and Setters

```
public class Car {
    private String owner;
    private String model;
    private int age;

    public void setOwner(String owner) {
        this.owner = owner;
    }

    public String getOwner() {
        return owner;
    }

    …
}
```

Factory Pattern

- Remove instantiation of actual implementation classes from client code

4 Fundamental Concepts

- **Inheritance:** one class based on another
- **Polymorphism:** single object -> many forms
- **Abstraction:** provide generalization
- **Encapsulation:** hide implementation details

OO Design

Parkin

ParkingLot

- ParkingLot
- Slot - small, large, etc
- Vehicle - Car, Bus, etc
- Parker?

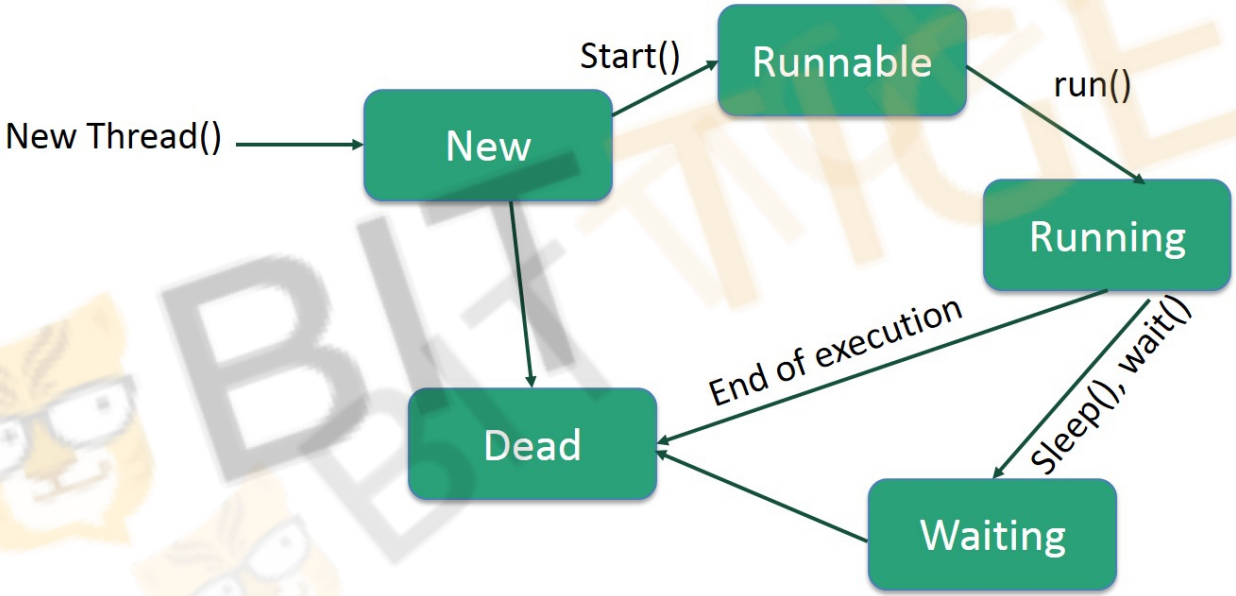
- Park car
- UnPark car
- Find empty space
- Is full?

Java Multithreading Basics

Thread vs Process

- A process has one or more threads
- Threads share the address space; processes have own address space
- Cancel, change priority, etc to the main thread may affect the behavior of other threads; changes to parent process does not affect child process

Life-cycle of a thread



Create a Thread

```
public class VehicleClient implements Runnable {
  @Override
  public void run() {
    builder.createVehicle("Car", "Bob", id);
  }
 }

new Thread(new VehicleClient()).start();
```

```
public class VehicleClient extends Thread {

  @Override
  public void run() {
  }
 }

new VehicleClient().start();
```

Ideal Case...

Thread 0: start createVehicle for “Bob” -> success

Thread 1: start createVehicle for “Bob” -> fail with duplicated owner

Thread 1 should not create successfully since we don’t allow duplicated owner.

Error!!!

Thread 0: Check “Bob” in the map? -> No -> Can add “Bob”

Thread 1: Check “Bob” in the map? -> No -> Can add “Bob”

Thread 1: Add “Bob”

Thread 0: Add “Bob”

Time

Keyword -

- Synchronized method
- Synchronized block

Synchronized

Synchronized Method

- No multiple invocations of synchronized methods on the same object to interleave.
- Changes to the state of the object are visible to all threads

```
synchronized void addVehicle(...) {

   // code ...

}
```

Synchronized Block

- Perform synchronization on any specific resource

```
synchronized (object reference expression) {

    // code ...

}
```

Bad design...

- Threads trying to access same data, but need to wait for each other
- Bouncing data from one thread to another
- Switch control of CPU

Generics

Generics

- Operate on various types
- Provide compile time safety

Generics

- Class Level Generics
- Method Generics
- Bounded Types
- Wildcards

Raw Type

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}


// No check for putting things in this box!
```


https://docs.oracle.com/javase/tutorial/java/generics/types.html

Generics

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

T: Type
E: Element
K: Key
V: Value

https://docs.oracle.com/javase/tutorial/java/generics/types.html

Raw Type vs Generics

```
Box b = new Box();              // raw type
b.set("Hello");

Integer c = (Integer) b.get();          // no compile error, runtime error
                                        // no warning even before 1.5
```

```
Box<String> b = new Box<>();
b.set("Hello");

Integer c = (Integer) b.get();          // compiler will complain!
```


https://docs.oracle.com/javase/tutorial/java/generics/types.html

Generic Classes

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
                this.key = key;
                this.value = value;
    }
    ….
}
```


https://docs.oracle.com/javase/tutorial/java/generics/types.html

Generic Methods

```
public class Util {
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}
```

Generic Methods

```
public class Util {
    public static <T extends Comparable<T>> boolean compare(T p1, T p2) {
        return p1.compareTo(p2);
    }
}
```

Bounded Types

```
public class Box<T extends Number> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Bounded Types

```
public class Box<T extends Number & String & List> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Bounded Types

```
public class Box<T super Warriors> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Wildcards

```
public static void process(List<? extends Foo> list) {
    for (Foo elem : list) {
        // ...
    }
}

public static void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}
```

Syllabus & Timeline

4.8	4.14	4.15
Data Structure	**Algorithm**	**Project & Java 8 & Java 9**
- Array - List - Map - Tree - Stack/Heap	- Binary Search - Sorting - BFS & DFS - DP	- Project - Lambda - Functional Interface - Stream API