Below are concise example code snippets for the 100 Java + Spring Boot interview questions. For purely conceptual items I either show a small illustrative snippet or a comment explaining what to run.

---

1. JDK / JRE / JVM

// No runnable code — conceptual only.

---

2. OOP examples (Encapsulation / Inheritance / Polymorphism / Abstraction)

```
// Encapsulation
public class BankAccount {
  private double balance;
  public BankAccount(double b){ this.balance = b; }
  public synchronized void deposit(double a){ balance += a; }
  public synchronized boolean withdraw(double a){ if(balance>=a){ balance-=a;
return true;} return false; }
  public double getBalance(){ return balance; }
}

// Inheritance & Polymorphism
abstract class Shape { abstract double area(); }
class Circle extends Shape { double r; Circle(double r){this.r=r;} double area()
{ return Math.PI*r*r;} }
class Square extends Shape { double s; Square(double s){this.s=s;} double area()
{ return s*s; } }

public class Demo {
  public static void main(String[] args){
    Shape s = new Circle(2.0); // runtime polymorphism
    System.out.println(s.area());
  }
}
```

---

3. Overloading vs Overriding

```
class A{
  void foo(int x){}
  void foo(String s){} // overloaded
```

```
}
class B extends A{
  @Override
  void foo(int x){} // overridden
}
```

---

4. private/static/final methods behavior

```
class Parent{
  private void p(){ System.out.println("private"); }
  static void s(){ System.out.println("static"); }
  final void f(){ System.out.println("final"); }
}
class Child extends Parent{
  // this is not overriding p(), just a new method
  private void p(){ System.out.println("child private"); }
  static void s(){ System.out.println("child static"); } // hides
  // cannot override final f()
}
```

---

5. Constructors

```
class Example{
  private int x;
  public Example(){ this(0); }
  public Example(int x){ this.x = x; }
}
```

---

6. Immutable class

```
public final class Person{
  private final String name;
  private final int age;
  public Person(String name,int age){ this.name = name; this.age = age; }
  public String getName(){ return name; }
  public int getAge(){ return age; }
}
```

7. String pool example

```
String a = "hello";
String b = "hello"; // same ref as a
String c = new String("hello"); // different object
System.out.println(a==b); // true
System.out.println(a==c); // false
```

---

8. String vs StringBuilder vs StringBuffer

```
String s = "a";
s += "b"; // creates new Strings

StringBuilder sb = new StringBuilder();
sb.append("a").append("b");

StringBuffer sbuf = new StringBuffer();
sbuf.append("a").append("b"); // synchronized
```

---

9. == vs equals

```
Integer i1 = 127; Integer i2 = 127; // cached
Integer i3 = 128; Integer i4 = 128; // not cached
System.out.println(i1==i2); // true
System.out.println(i3==i4); // false
System.out.println(i3.equals(i4)); // true
```

---

10. hashCode() and equals()

```
class Person{
  String id;
  Person(String id){this.id=id;}
  @Override public boolean equals(Object o){
    if(this==o) return true;
    if(o==null||getClass()!=o.getClass()) return false;
    return id.equals(((Person)o).id);
```

```
  }
  @Override public int hashCode(){ return id.hashCode(); }
}
```

---

11. Marker interface example

```
// Serializable is a marker interface - no code example needed beyond marking
class:
public class Data implements java.io.Serializable { private int x; }
```

---

12. Abstract class vs Interface

```
abstract class Animal{ abstract void speak(); }
interface Pet{ void play(); default void wash(){
System.out.println("washing"); } }
class Dog extends Animal implements Pet{ void speak(){
System.out.println("woof"); } public void play(){} }
```

---

13. Default methods in interfaces

```
interface I{ default void hello(){ System.out.println("hi"); } }
```

14–16. Lambdas, Streams, Optional

```
import java.util.*;
public class FP {
  public static void main(String[] args){
    List<String> names = Arrays.asList("alice","bob","carl");
    names.stream().filter(s-
>s.startsWith("a")).map(String::toUpperCase).forEach(System.out::println);

    Optional<String> maybe = Optional.ofNullable(null);
    System.out.println(maybe.orElse("default"));
  }
}
```

17. Optional usage

```
Optional<String> getName(){ return Optional.ofNullable(name); }
```

18–19. Exceptions and try-with-resources

```
try (java.io.InputStream in = new java.io.FileInputStream("file.txt")) {
  // use stream
} catch (IOException e){ e.printStackTrace(); }
```

20–21. Generics and type erasure

```
public class Box<T>{ private T t; public void set(T t){ this.t = t; } public T
get(){ return t; }}
// type erasure means at runtime Box<String> and Box<Integer> are same class
```

---

22. final/finally/finalize

```
final class F {}
try { } finally { /* always runs */ }
// finalize is deprecated; don't use
```

23–25. JMM, GC, ClassLoader

// Conceptual — code: show volatile example

```
class Flag {
  private volatile boolean stop = false;
  public void requestStop(){ stop = true; }
  public void loop(){ while(!stop) { /* busy work */ } }
}
```

26–40 Collections examples

```
// ArrayList vs LinkedList
List<String> al = new ArrayList<>();
List<String> ll = new LinkedList<>();

// HashMap basic
Map<String,Integer> m = new HashMap<>(); m.put("a",1);

// ConcurrentHashMap
ConcurrentHashMap<String,Integer> cm = new ConcurrentHashMap<>();

// CopyOnWriteArrayList
CopyOnWriteArrayList<Integer> cow = new CopyOnWriteArrayList<>();

// PriorityQueue
PriorityQueue<Integer> pq = new PriorityQueue<>(); pq.add(5); pq.add(1);

// Comparator vs Comparable
class Person implements Comparable<Person>{ String name; public int
compareTo(Person o){ return name.compareTo(o.name);} }
Comparator<Person> byName = Comparator.comparing(p->p.name);

// WeakHashMap
Map<Object,String> weak = new WeakHashMap<>();
```

41–60 Concurrency examples

```
// Runnable vs Callable
Runnable r = ()-> System.out.println("run");
Callable<String> c = ()-> "done";

// ExecutorService
ExecutorService exec = Executors.newFixedThreadPool(4);
Future<String> f = exec.submit(c);

// ReentrantLock
ReentrantLock lock = new ReentrantLock();
lock.lock(); try { /* critical */ } finally { lock.unlock(); }

// CountDownLatch
CountDownLatch latch = new CountDownLatch(3);
// Semaphore
```

```
Semaphore sem = new Semaphore(2);

// CompletableFuture
CompletableFuture.supplyAsync(() -> 42).thenApply(i-
>i*2).thenAccept(System.out::println);

// ThreadLocal
ThreadLocal<DateFormat> tl = ThreadLocal.withInitial(() -> new
SimpleDateFormat("yyyy-MM-dd"));

// ForkJoin sample
class SumTask extends RecursiveTask<Long> {
  private long[] arr; int lo,hi; long TH = 1000;
  SumTask(long[] arr, int lo,int hi){ this.arr=arr;this.lo=lo;this.hi=hi; }
  protected Long compute(){ if(hi-lo<TH){ long s=0; for(int i=lo;i<hi;i++)
s+=arr[i]; return s; } else { int mid=(lo+hi)/2; SumTask l=new
SumTask(arr,lo,mid); SumTask r=new SumTask(arr,mid,hi); l.fork(); long right =
r.compute(); long left = l.join(); return left+right; }}
}
```

61–70 JVM / Performance snippets

```
// Example: print thread dump
Thread.getAllStackTraces().forEach((t, st) -> System.out.println(t.getName()));

// Example: request GC (not guaranteed)
System.gc();
```

71–80 Spring Core examples

```
// Spring Boot minimal app
// build.gradle: include 'org.springframework.boot:spring-boot-starter'
@SpringBootApplication
public class App { public static void main(String[] args){
SpringApplication.run(App.class,args);} }

// Component and injection
@Component class MyService{ public String greet(){ return "hi"; } }
@RestController class C{ private final MyService svc; public C(MyService s)
{this.svc=s;} @GetMapping("/g") String g(){ return svc.greet(); } }
```

81–90 Spring Boot examples

```
// application.yml example
// server:
//   port: 8081

// Actuator (add dependency) exposes /actuator/health

// @ControllerAdvice
@ControllerAdvice
class Global{ @ExceptionHandler(Exception.class) ResponseEntity<String>
e(Exception ex){ return ResponseEntity.status(500).body(ex.getMessage()); } }

// JWT filter (simplified)
public class JwtFilter extends OncePerRequestFilter{
  protected void doFilterInternal(HttpServletRequest req, HttpServletResponse
res, FilterChain chain) throws ServletException, IOException{
    String token = req.getHeader("Authorization"); // validate token
    chain.doFilter(req,res);
  }
}
```

91–100 Microservices examples & patterns

```
// RestTemplate call (sync)
RestTemplate rt = new RestTemplate(); String body = rt.getForObject("http://
usersvc/api/1", String.class);

// Feign client (declarative)
@FeignClient(name="users") interface UserClient{ @GetMapping("/api/{id}") User
get(@PathVariable String id); }

// Kafka producer (spring-kafka)
@Autowired KafkaTemplate<String,String> kt; kt.send("topic","key","msg");

// Circuit Breaker with Resilience4j
Supplier<String> decorated = CircuitBreaker.decorateSupplier(cb, ()->
restTemplate.getForObject("http://svc/hi", String.class));

// Distributed tracing: Spring Cloud Sleuth adds trace ids to logs
(configuration only)

// Idempotency example (pseudo)
```

```
@PostMapping
public ResponseEntity<?> create(@RequestHeader("Idempotency-Key") String key){
if(alreadyProcessed(key)) return ResponseEntity.status(200).build(); process();
markProcessed(key); }

// Spring Cloud Config: use config server URL in bootstrap properties
```

Notes

- These snippets are intentionally short; expand them into full projects when practicing (for instance: build a Spring Boot app with controllers, services, tests).
- For multi-threading and concurrency, writing small unit tests and reproducing race conditions/locks in sample programs will improve understanding.

If you'd like, I can now: - Convert selected snippets into runnable mini-projects (e.g., a Spring Boot microservice with JWT, Kafka producer/consumer, Dockerfile, Kubernetes manifest). - Or expand any specific question into a full, annotated code sample with tests and explanation.

Which one should I expand into a runnable project first?