QUINBAY ASSIGNMENT

PHASE 1: LEARNING JAVA – PRE-READ

HEAD FIRST JAVA – A BRAIN FRIENDLY GUIDE

Submitted by:

Jyothsna K

CHAPTER 1 – BREAKING THE SURFACE

SHARPEN YOUR PENCIL(Pg-5)

Q) The naming conventions for Java's versions are confusing. There was JDK 1.0, and 1.2, 1.3, 1.4, then a jump to J2SE 5.0, then it changed to Java 6, Java 7, and last time I checked, Java was up to Java 18. What's going on?

Ans- There are various versions of java available. There is a jump from Java 1.4 to Java 5.0, but Java 5.0 is the same as Java 1.5. The "1." has been dropped and the versions are named similarly. Therefore Java 6 is same as Java 1.6, Java 7 is same as Java 1.7 and so on. However, since this naming was confusing, it has been changed from Java 9. This means that Java 9 does not represent Java 1.9. This holds true for all the future versions. All these versions are backward compatible which means that a program written for Java 5 will run on a Java 8 or Java 11 environment without modification. Java releases a new version every 6 months, leading to the release of Java 18.

Q) Interpret the meaning of each line of code

int size = 27;	→ Declare an integer variable named 'size' and give it the value 27
String name = "Fido";	→ Declare a String variable named 'name' and give it the value "Fido"
Dog myDog = new Dog(name, size);	→ Declare an object of Class Dog, named 'myDog', with attributes name and size
x = size - 5;	→ Variable 'x' is assigned value size-5, ie 27-5=22
if (x < 15) myDog.bark(8);	→ If x<15, ie 22<15 (false) then bark() method of Dog class is called (condition is false, so not executed)
while (x > 3)	→ While x(ie, 22) is more than 3, code block is executed

{ myDog.play(); }	→ The play() method of Dog class is called
int[] numList = {2, 4, 6, 8};	→ Declare an integer array called 'numList' with values = {2,4,6,8}
System.out.print("Hello");	→print "Hello"
System.out.print("Dog: " + name);	→print "Dog: Fido"
String num = "8";	→ Declare a character string variable 'num' and give it the value of "8"
int z = Integer.parseInt(num);	→ convert the string 'num' ("8") into numeric value 8
try { readTheFile("myFile.txt"); }	→trying to read 'myFile.txt' file
catch (FileNotFoundException ex)	→catching an exception
{ System.out.print("File not found."); }	→printing "File not found" if exception is caught

THERE ARE NO DUMB QUESTIONS(Pg 14)

Q) Why does everything have to be in a class?

Ans- Java is an object-oriented programming language. A java code mostly consists of object. Since class is a blueprint of an object, everything has to be inside a class.

Q) Do I have to put a main in every class I write?

Ans- It is not necessary that every class should have a main method, but it is essential that at least one class must have a main method. The program starts getting executed from the main method.

Q) In my other language I can do a boolean test on an integer. In Java, can I say something like:

```
int x = 1;

while (x){ }
```

Ans- No, since integer and Boolean are not compatible, the above statement is invalid. It will not compile due to type mismatch error. We can use a Boolean variable instead.

SHARPEN YOUR PENCIL(Pg 15)

```
public class DooBee{
        public static void main(String args[]){
                int x=1;
                while(x<3){
                        System.out.print(“Doo”);
                        System.out.print(“Bee”);
                        x=x+1;
                }
        If(x==3){
                System.out.println(“Do”);
}
}
}
```

CODE MAGNETS (Pg 20)

```
class Shuffle1 {

  public static void main(String[] args) {

    int x = 3;

    while (x > 0) {

      if (x > 2) {

        System.out.print("a");

      }

      x = x - 1;

      System.out.print("-");

      if (x == 2) {
```

```
      System.out.print("b c");
    }
    if (x == 1) {
      System.out.print("d");
      x = x - 1;
    }
  }
 }
}
```

BE THE COMPILER (Pg 21):

A

```
class Exercise1a {
 public static void main(String[] args) {
 int x = 1;
 while (x < 10) {
 if (x > 3) {
 System.out.println("big x");
 }
 }
 }
 }
```

Ans: The file will compile but will be an infinite loop, so we add an increment statement x++; right after closing the if-block.

OUTPUT:

big x

big x

big x

big x

big x

big x

B

```
public static void main(String [] args) {
 int x = 5;
 while ( x > 1 ) {
 x = x - 1;
 if ( x < 3) {
 System.out.println("small x");
 }
 }
}
```

Ans: Code wont compile because no class is defined, so the main method needs to be added inside a class (ex: Exercise1b)

OUTPUT:

small x

small x

C

```
class Exercise1c {
 int x = 5;
 while (x > 1) {
 x = x - 1;
 if (x < 3) {
 System.out.println("small x");
 }
 }
 }
```

Ans: Code wont compile because no main method is defined for this class. So the code block inside class must be inside a main method.

OUTPUT:

small x

small x

CHAPTER 2 – A TRIP TO OBJECTVILLE

SHARPEN YOUR PENCILS (Pg 37)

The MovieTestDrive class creates objects (instances) of the Movie class and uses the dot operator (.) to set the instance variables to a specific value. The MovieTestDrive class also invokes (calls) a method on one of the objects. Fill in the chart to the right with the values the three objects have at the end of main().

object 1	title	-	Gone with the Stock
	genre	-	Tragic
	rating	-	-2
object 2	title	-	Lost in Cubicle Space
	genre	-	Comedy
	rating	-	5
object 3	title	-	Byte Club
	genre	-	Tragic but ultimately uplifting
	rating	-	127

THERE ARE NO DUMB QUESTIONS(Pg 41)

Q. What if I need global variables and methods? How do I do that if everything has to go in a class?

Ans: In Java, if you need global variables and methods, you can achieve this by using static or public variables and methods within a class. Static variables and methods belong to the class itself rather than to instances of the class. We can also use public, static and final to make a global constant.

Q. Then how is this object oriented if you can still make global functions and global data?

Ans: Java remains object-oriented as static members are tied to classes, not instances, enforcing encapsulation, public and static methods can access only static variables and methods. This ensures object-oriented principles are maintained while allowing class-level global access.

Q. What is a Java program? What do you actually deliver?

Ans: A Java program is a collection of classes and methods compiled into bytecode. The deliverable is a compiled .class file or a packaged .jar file, which can be executed on the Java Virtual Machine (JVM).

Q. What if I have a hundred classes? Or a thousand? Isn't that a big pain to deliver all those individual files? Can I bundle them into one Application Thing?

Ans: Yes, we can bundle multiple classes into a single Java ARchive (JAR) file. This JAR file contains all the compiled .class files and resources, making it easier to deliver and manage our application.

BE THE COMPILER (Pg 42)

```
A. class StreamingSong {
 String title;
 String artist;
 int duration;
 void play() {
 System.out.println("Playing song");
 }
 void printDetails() {
 System.out.println("This is " + title + " by " + artist);
 }
}
class StreamingSongTestDrive {
 public static void main(String[] args) {
StreamingSong song = new StreamingSong();
 song.artist = "The Beatles";
 song.title = "Come Together";
```

```
song.play();
song.printDetails();
}
}
```

B.

```
class Episode {
int seriesNumber;
int episodeNumber;
void play() {
System.out.println("Playing episode " + episodeNumber);
}
void skipIntro() {
System.out.println("Skipping intro...");
}
void skipToNext() {
System.out.println("Loading next episode...");
}
}
class EpisodeTestDrive {
public static void main(String[] args) {
Episode episode = new Episode();
```

```
episode.seriesNumber = 4;

episode.play();

episode.skipIntro();

}

}
```

CODE MAGNETS (Pg 43)

Q. A Java program is all scrambled up on the fridge. Can you reconstruct the code snippets to make a working Java program that produces the output listed below? Some of the curly braces fell on the floor and they were too small to pick up, so feel free to add as many of those as you need.

Ans:

```
class DrumKit {

 boolean topHat = true;

 boolean snare = true;

 void playTopHat() {

 System.out.println("ding ding da-ding");

 }

 void playSnare() {

 System.out.println("bang bang ba-bang");

 }
```

```
}

class DrumKitTestDrive {

 public static void main(String[] args) {

 DrumKit d = new DrumKit();

 d.playSnare();

 d.snare = false;

 d.playTopHat();

 if (d.snare == true) {

 d.playSnare();

 }

 }

 }
```

CHAPTER 3 – BREAKING THE SURFACE

SHARPEN YOUR PENCIL

Pg 52

1. int x = 34.5;
2. boolean boo = x;
3. int g = 17;
4. int y = g;
5. y = y+ 10;
6. short s;
7. s = y;
8. byte b = 3;
9. byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;

Legal statements in the above ones are:

Statements 3, 4, 5, 6, and 10.

Pg 60

Q) What is the current value of pets[2]?

Ans – null, as pets[2] does not refer to any object currently.

Q) What code would make pets[3] refer to one of the two existing Dog objects?

Ans – This assigns the reference of Dog1 (currently referred to by pets[1]) to pets[3].

```
pets[3] = pets[1];
```

THERE ARE NO DUMB QUESTIONS(Pg 56)

Q) How big is a reference variable?

Ans- The exact size isn't directly accessible, but imagine it as a 64-bit value.

Q) So, does that mean that all object references are the same size, regardless of the size of the actual objects to which they refer?

Ans- Yes, within a JVM, references have the same size.

Q) Can I do arithmetic on a reference variable, increment it, you know—C stuff?

Ans- No, Java restricts direct pointer arithmetic.

BE THE COMPILER (Pg 63)

A

```
class Books {
  String title;
  String author;
 }
 class BooksTestDrive {
  public static void main(String[] args) {
```

```
  Books[] myBooks = new Books[3];
  int x = 0;
  myBooks[0].title = "The Grapes of Java";
  myBooks[1].title = "The Java Gatsby";
  myBooks[2].title = "The Java Cookbook";
  myBooks[0].author = "bob";
  myBooks[1].author = "sue";
  myBooks[2].author = "ian";
   while (x < 3) {
    System.out.print(myBooks[x].title);
    System.out.print(" by ");
    System.out.println(myBooks[x].author);
    x = x + 1;
  }
 }
}
```

Ans:

Issue: The myBooks array is declared but not initialized. Each element in the array is still null.

Solution: Initialize each element of the myBooks array by creating new Books objects.

```
myBooks[0] = new Books();
myBooks[1] = new Books();
myBooks[2] = new Books();
```

B

```
class Hobbits {
 String name;
 public static void main(String[] args) {
   Hobbits[] h = new Hobbits[3];
   int z = 0;
   while (z < 3) {
     z = z + 1;
     h[z] = new Hobbits();
     h[z].name = "bilbo";
     if (z == 1) {
       h[z].name = "frodo";
     }
     if (z == 2) {
       h[z].name = "sam";
     }
     System.out.print(h[z].name + " is a ");
     System.out.println("good Hobbit name");
   }
 }
}
```

Ans:

Issue: The loop condition while (z < 4) will cause an ArrayIndexOutOfBoundsException because the array h has a length of 3 (indices 0, 1, and 2).

Solution: Change the loop condition to while (z < 3) to stay within the valid array indices.

CODE MAGNETS (Pg 64)

```
class TestArrays {

public static void main(String [] args) {

String [] islands = new String[4];

   int[] index = new int[4];

   index[0] = 1;

   index[1] = 3;

   index[2] = 0;

   index[3] = 2;

   String[] islands = new String[4];

   islands[0] = "Bermuda";

   islands[1] = "Fiji";

   islands[2] = "Azores";

   islands[3] = "Cozumel";

   int y = 0;

   int ref;

   while (y < 4) {

     ref = index[y];

     System.out.print("island = ");
```

```
    System.out.println(islands[ref]);
    y = y + 1;
  }
 }
}
```

CHAPTER 4 – HOW OBJECTS BEHAVE

THERE ARE NO DUMB QUESTIONS(Pg: 78)

Q: **What happens if the argument you want to pass is an object instead of a primitive?**

Ans: If the argument you want to pass is an object instead of a primitive, the object reference is passed, allowing the method to modify the object's state.

Q: **Can a method declare multiple return values? Or is there some way to return more than one value?**

Ans: No, a method in Java can only return a single value. Yes, you can return more than one value by using objects, arrays, or collections (like lists or maps).

Q: **Do I have to return the exact type I declared?**

Ans: We can return anything that can be implicitly promoted to that type. We pass a byte where an int is expected.

Q: **Do I have to do something with the return value of a method? Can I just ignore it?**

Ans: Java doesn't require you to acknowledge a return value .In Java, you don't have to assign or use the return value.

Pg: 85

Q: What about method parameters? How do the rules about local variables apply to them?

Ans: Method parameters are virtually the same as local variables—they're declared *inside* the method. But method parameters will never be uninitialized, so you'll never get a compiler error telling you that a parameter variable might not have been initialized. Instead, the compiler will give you an error if you try to invoke a method without giving the arguments that the method needs.

SHARPEN YOUR PENCIL (Pg: 87)

Q) Given the method below, which of the method calls listed on the right are legal?

```
int calcArea(int height, int width) {
return height * width;
}
int a = calcArea(7, 12);
short c = 7;
calcArea(c, 15);
int d = calcArea(57);
calcArea(2, 3);
long t = 42;
int f = calcArea(t, 17);
int g = calcArea();
calcArea();
byte h = calcArea(4, 20);
int j = calcArea(2, 3, 5);
```

Ans:

The legal method calls are-

i) int a = calcArea(7, 12);

ii) calcArea(2, 3);

BE THE COMPILER(Pg: 88)

Q) Each of the Java files on this page represents a complete source file. Your job is to play compiler and determine whether each of these files will compile. If they won't compile, how would you fix them, and if they do compile, what would be their output?

Code A:

```
class XCopy {

public static void main(String[] args) {

int orig = 42;

XCopy x = new XCopy();

int y = x.go(orig);

System.out.println(orig + " " + y);

}

int go(int arg) {

arg = arg * 2;

return arg;

}
```

Ans: The above code gets compiled

Output: 42 84

Code B:

```
class Clock {

String time;

void setTime(String t) {

time = t;

}

void getTime() {

return time;

}

}

class ClockTestDrive {

public static void main(String[] args) {

Clock c = new Clock();

c.setTime("1245");

String tod = c.getTime();

System.out.println("time: "+tod);

}

}
```

Ans: The above code will not compile because the method getTime returns a value but is declared as void in the function header. The code will work if the return type is declared as String

Corrected Code:

```
class Clock {

String time;
```

```
void setTime(String t) {
time = t;
}
String getTime() {
return time;
}
}
class ClockTestDrive {
public static void main(String[] args) {
Clock c = new Clock();
c.setTime("1245");
String tod = c.getTime();
System.out.println("time: "+tod);
}
}
```

EXERCISE: WHO AM I (Pg 89)

Q) Fill in the blanks next to the sentence with the names of one or more attendees.

Tonight's attendees:

instance variable, argument, return, getter, setter,

encapsulation, public, private, pass by value, method

Ans:

A class can have any number of these → instance variables, getter, setter, method

A method can have only one of these.	→	return
This can be implicitly promoted.	→	return, argument
I prefer my instance variables private.	→	encapsulation
It really means “make a copy.”	→	pass by value
Only setters should update these.	→	instance variables
A method can have many of these.	→	argument
I return something by definition.	→	getter
I shouldn’t be used with instance variables.	→	public
I can have many arguments.	→	method
By definition, I take one argument.	→	setter
These help create encapsulation.	→	getter, setter, public, private
I always fly solo.	→	return

CHAPTER 5 – EXTRA-STRENGTH METHODS

SHARPEN YOUR PENCIL(Pg 107)

Q. We built the test class and the SimpleStartup class. But we still haven't made the actual game. Given the code on the opposite page and the spec for the actual game, write in your ideas for prep code for the game class. We've given you a few lines here and there to get you started.

Ans:

```
public static void main(String[] args) {
int numOfGuesses = 0;
SimpleStartup startup = new SimpleStartup();
int startPos = (int) (Math.random() * 5);
int[] locations = {startPos, startPos + 1, startPos + 2};
startup.setLocationCells(locations);
boolean isAlive = true;
Scanner scanner = new Scanner(System.in);
while (isAlive) {
    System.out.print("Enter a guess: ");
    String userGuess = scanner.nextLine();
    String result = startup.checkYourself(userGuess);
    numOfGuesses++;
```

```
        if (result.equals("kill")) {
            isAlive = false;
            System.out.println("You took " + numOfGuesses + " guesses");
        }
    }
     scanner.close();
}
```

BE THE JVM(pg 118)

Q. class Output {

public static void main(String[] args) {

Output output = new Output();

output.go();

}

void go() {

int value = 7;

for (int i = 1; i < 8; i++) {

value++;

if (i > 4) {

System.out.print(++value + " ");

}

```
if (value > 14) {
System.out.println(" i = " + i);
break;
} }
}
}
```

Ans: Output:

13 15 x = 6

CODE MAGNETS(Pg 119)

Q. A working Java program is all scrambled up on the fridge. Can you reconstruct the code snippets to make a working Java program that produces the output listed below? Some of the curly braces fell on the floor and they were too small to pick up, so feel free to add as many of those as you need!

Ans:

```
class MultiFor {
public static void main(String[] args) {
for (int i = 0; i < 4; i++) {
for (int j = 4; j > 2; j--) {
System.out.println(i + " " + j);
}
if (i == 1) {
i++;
} }
} }
```

CHAPTER 6 – USING THE JAVA LIBRARY

SHARPEN YOUR PENCIL(Pg 134)

Q.

ArrayList<String> myList = new ArrayList<String>();	→ String [] myList = new String[2];
String a = "whoohoo";	→ String a = "whoohoo";
myList.add(a);	→ myList[0] = a;
String b = "Frog";	→ String b = "Frog";
myList.add(b);	→ myList[1] = b;
int theSize = myList.size();	→ int theSize = myList.length;
String str = myList.get(1);	→ String str = myList[1];
myList.remove(1);	→ myList[1] = null;

boolean isIn = myList.contains(b);	→ boolean isIn = false;
	for (String item : myList) {
	if (b.equals(item)) {
	isIn = true;
	break;
	}
	}

Pg 146

Q. Annotate the code yourself! Match the annotations at the bottom of each page with the numbers in the code. Write the number in the slot in front of the corresponding annotation. You'll use each annotation just once, and you'll need all of the annotations.

```
import java.util.ArrayList;

public class StartupBust {
  private GameHelper helper = new GameHelper();
  private ArrayList<Startup> startups = new ArrayList<Startup>();
  private int numOfGuesses = 0;

  private void setUpGame() {
    // first make some Startups and give them locations
    Startup one = new Startup();
    one.setName("poniez");
    Startup two = new Startup();
    two.setName("hacqi");
    Startup three = new Startup();
    three.setName("cabista");
    startups.add(one);
    startups.add(two);
    startups.add(three);

    System.out.println("Your goal is to sink three Startups.");
    System.out.println("poniez, hacqi, cabista");
    System.out.println("Try to sink them all in the fewest number of guesses");

    for (Startup startup : startups) {
      ArrayList<String> newLocation = helper.placeStartup(3);
      startup.setLocationCells(newLocation);
    } // close for loop
  } // close setUpGame method

  private void startPlaying() {
    while (!startups.isEmpty()) {
      String userGuess = helper.getUserInput("Enter a guess");
      checkUserGuess(userGuess);
    } // close while
    finishGame();
  } // close startPlaying method
```

1
2
3
4
5
6
7
8
9
10

```
  private void checkUserGuess(String userGuess) {
    numOfGuesses++;  (11)
    String result = "miss";  (12)

    for (Startup startupToTest : startups) {  (13)
      result = startupToTest.checkYourself(userGuess);  (14)

      if (result.equals("hit")) {
        break;  (15)
      }
      if (result.equals("kill")) {
        startups.remove(startupToTest);  (16)
        break;
      }
    } // close for

    System.out.println(result);  (17)
  } // close method


  private void finishGame() {
    System.out.println("All Startups are dead! Your stock is now worthless");
    if (numOfGuesses <= 18) {
      System.out.println("It only took you " + numOfGuesses + " guesses.");
      System.out.println("You got out before your options sank.");
    } else {
      System.out.println("Took you long enough. " + numOfGuesses + " guesses.");
      System.out.println("Fish are dancing with your options");
    }
  } // close method  (18)

  public static void main(String[] args) {
    StartupBust game = new StartupBust();  (19)
    game.setUpGame();  (20)
    game.startPlaying();  (21)
  } // close method
}
```

Ans:

1→ Declare and initialize the variables we'll need.

2→ Make three Startup objects, give 'em names, and stick 'em in the ArrayList

3→ Print brief instructions for user.

4→ Repeat with each Startup in the list.

5→ Ask the helper for a Startup location (an ArrayList of Strings).

6→ Call the setter method on this Startup to give it the location you just got from the helper.

7→ As long as the Startup list is NOT empty (the ! means NOT, it's the same as

(startups.isEmpty() == false).

8→ Get user input.

9→ Call our own checkUserGuess method.

10→ Call our own finishGame method.

11→ Increment the number of guesses the user has made

12→ Assume it's a 'miss', unless told otherwise

13→ Repeat with all Startups in the list

14→ Ask the Startup to check the user guess, looking for a hit (or kill)

15→ Get out of the loop early, no point in testing the others

16→ This one's dead, so take it out of the Startups list then get out of the loop

17→ Print the result for the user

18→ Print a message telling the user how they did in the game

19→ Create the game object

20→ Tell the game object to set up the game

21→ Tell the game object to start the main game play loop (keeps asking for user input and checking the guess)

THERE ARE NO DUMB QUESTIONS(pg 156)

Q. Why does there have to be a full name? Is that the only purpose of a package?

Ans: A full package name ensures unique class names across different projects. This avoids naming conflicts and organizes related classes together. A package groups related classes and interfaces together, providing modularity and a namespace to avoid naming conflicts. It also helps in organizing code and managing access control.

Q. OK, back to the name collision thing. How does a full name really help? What's to prevent two people from giving a class the same package name?

Ans: A full package name helps by typically including a unique domain name owned by the developer (e.g., com.example.project). This convention reduces the likelihood of name collisions since domain names are globally unique.

Q. Does import make my class bigger? Does it actually compile the imported class or package into my code?

Ans: No, the import statement does not make your class bigger. It does not compile the imported class or package into your code; it simply tells the compiler where to find the referenced classes.

Q. OK, how come I never had to import the String class? Or System?

Ans: The String and System classes are part of the java.lang package, which is automatically imported by default in every Java program.

Q. Do I have to put my own classes into packages? How do I do that? Can I do that?

Ans: You don't have to put your classes into packages, but it's good practice for organization and avoiding naming conflicts. To do this, add a package statement at the top of your Java file.

CODE MAGNETS(pg 163)

Q. Can you reconstruct the code snippets to make a working Java program that produces the output listed below? NOTE: To do this exercise, you need one NEW piece of info—if you look in the API for ArrayList, you'll find a second add method that takes two arguments:

add(int index, Object o)

Ans: import java.util.ArrayList;

```
public class ArrayListMagnet {

 public static void main(String[] args) {

 ArrayList<String> a = new ArrayList<String>();

 a.add(0, "zero");

 a.add(1, "one");

 a.add(2, "two");

 a.add(3, "three");

 printList(a);

 if (a.contains("three")) {

 a.add("four");

 }

 a.remove(2);

 printList(a);

 if (a.indexOf("four") != 4) {

 a.add(4, "4.2");

 }

 printList(a);

 if (a.contains("two")) {

 a.add("2.2");

 }

 printList(a);

 }

 public static void printList(ArrayList<String> list) {
```

```
 for (String element : list) {
 System.out.print(element + " ");
 }
 System.out.println();
 }
}
```

CHAPTER 7 – BETTER LIVING IN OBJECTVILLE

SHARPEN YOUR PENCIL(pg 178)

Class	Superclasses	Subclasses
Musician	Artist	Singer
Rock Star	Musician	Lead Guitarist
Fan	Person	MusicFan
Bass Player	Instrumentalist	SessionPlayer
Concert Pianist	Pianist	Classical Pianist

THERE ARE NO DUMB QUESTIONS(Pg 178)

Q. You said that the JVM starts walking up the inheritance tree, starting at the class type you invoked the method on (like the Wolf example on the previous page). But what happens if the JVM doesn't ever find a match?

Ans: If the JVM doesn't find a matching method in the class or its superclasses, it will continue searching up the inheritance chain. If it reaches the top of the hierarchy without finding a match, it will throw a NoSuchMethodError at runtime. This indicates that the method you're trying to call does not exist. Ensure that the method name, parameters, and return type match exactly. If the

method is intended to be available, check the class hierarchy and method signatures for correctness.

SHARPEN YOUR PENCIL(pg 181)

Q. Put a check next to the relationships that make sense:

Oven extends Kitchen ❌

Guitar extends Instrument ✔

Person extends Employee ✔

Ferrari extends Engine ❌

FriedEgg extends Food ✔

Beagle extends Pet ✔

Container extends Jar ✔

Metal extends Titanium ❌

GratefulDead extends Band ✔

Blonde extends Smart ❌

Beverage extends Martini ❌

THERE ARE NO DUMB QUESTIONS

Pg 182

Q. So we see how a subclass gets to inherit a superclass method, but what if the superclass wants to use the subclass version of the method?

Ans: If the superclass wants to use the subclass version of a method, it can't directly call the subclass method from within the superclass. Instead, you can use method overriding, where the subclass provides its own implementation of the method. When the superclass method is called on an object of the subclass, the JVM dynamically dispatches the call to the subclass's overridden method.

Q. In a subclass, what if I want to use BOTH the superclass version and my overriding subclass version of a method? In other words, I don't want to completely replace the superclass version; I just want to add more stuff to it.

Ans: In a subclass, you can call the superclass version of a method by using super.methodName() within your overridden method. This allows you to add or modify functionality while retaining the behavior of the superclass method. For example, use super.methodName() at the beginning or end of your method to include the superclass's implementation and then add additional logic as needed.

Pg 191

Q. Are there any practical limits on the levels of subclassing? How deep can you go?

Ans: While Java doesn't enforce a strict limit on the number of subclass levels, practical limits are imposed by factors like code readability, maintainability, and JVM stack size. Deep subclass hierarchies can lead to complex and hard-to-manage code. Additionally, the JVM has a stack depth limit, which can be reached if method calls become excessively deep. Generally, it's best to keep class hierarchies reasonably shallow for clarity and ease of maintenance.

Q. Hey, I just thought of something...if you don't have access to the source code for a class but you want to change the way a method of that class works, could you use subclassing to do that? To extend the "bad" class and override the method with your own better code?

Ans: Yes, you can use subclassing to extend a class you don't have source code for and override its methods to change behavior. By creating a subclass and providing a new implementation for the method, you can modify its functionality while still using the original class's interface. However, this

only works if the method you want to override is not final, private, or static, as those cannot be overridden.

Q. Can you extend any class? Or is it like class members where if the class is private you can't inherit it.

Ans: In Java, you cannot extend a private class because private access is limited to the class itself. However, you can extend a protected or public class. If a class is protected or public, it can be inherited as long as it is accessible from your code. Additionally, classes that are not explicitly declared public have package-private visibility, meaning they can be extended only within the same package.

Q. Why would you ever want to make a final class? What advantage would there be in preventing a class from being subclassed?

Ans: Making a class final prevents it from being subclassed, which can be advantageous for ensuring security and integrity. It helps maintain consistent behavior and avoids unintended modifications or extensions that could introduce bugs. Additionally, final classes can improve performance, as the JVM can optimize method calls knowing the class won't change. This is particularly useful for utility classes or immutable objects.

Q. Can you make a method final, without making the whole class final?

Ans: Yes, you can make a method final without making the entire class final. Declaring a method as final prevents it from being overridden by subclasses, while the class itself can still be subclassed if it's not marked as final. This allows you to protect specific methods from modification while retaining the ability to extend the class.

BE THE COMPILER(Pg 195)

Q. public class MonsterTestDrive {

public static void main(String[] args) {

Monster[] monsters = new Monster[3];

monsters[0] = new Vampire();

monsters[1] = new Dragon();

monsters[2] = new Monster();

for (int i = 0; i < monsters.length; i++) {

monsters[i].frighten(i);

}

}

}

class Monster {

(A)

}

class Vampire extends Monster {

(B)

}

class Dragon extends Monster {

boolean frighten(int degree) {

System.out.println("breathe fire");

return true;

}

}

Ans:

A→ boolean frighten(int d) {

System.out.println("arrrgh");

return true;

}

B→ boolean frighten(int x) {

System.out.println("a bite?");

return false;

}

CHAPTER 8 – SERIOUS POLYMORPHISM

THERE ARE NO DUMB QUESTIONS(pg 205)

Q. What is the point of an abstract method? I thought the whole point of an abstract class was to have common code that could be inherited by subclasses.

Ans: An abstract method defines a contract for subclasses, specifying that they must provide their own implementation of the method. The point of an abstract method is to ensure that all subclasses fulfill certain behaviors while allowing flexibility in how those behaviors are implemented. An abstract class can still provide common code and default behavior, but abstract methods enforce that specific methods are implemented by any concrete subclass.

SHARPEN YOUR PENCIL(pg 207)

Q. Let's put all this abstract rhetoric into some concrete use. In the middle column we've listed some classes. Your job is to imagine applications where the listed class might be concrete, and applications where the listed class might be abstract. We took a shot at the first few to get you going. For example, class Tree would be abstract in a tree nursery program, where differences between an Oak and an Aspen matter. But in a golf simulation program, Tree might be a concrete class (perhaps a subclass of Obstacle), because the program doesn't care about or distinguish between different types of trees.

Ans:

Concrete	**Sample Class**	**Abstract**
golf course simulation	Tree	tree nursery application
architect application	House	architect application
satellite photo application	Town	satellite photo application
football coaching application	Football Player	coaching application

furniture store application	Chair	furniture design application
customer management system	Customer	customer service application
order processing system	Sales Order	order management application
bookstore inventory system	Book	library management application
retail store inventory system	Store	store management application
golf equipment shop	Golf Club	golf course simulation
automotive repair shop	Carburetor	automotive parts management
kitchen appliance store	Oven	appliance repair application

THERE ARE NO DUMB QUESTIONS(pg 212,227,230,231)

Q. Is class Object abstract?

Ans: No, the Object class in Java is not abstract; it is a concrete class. It is the root of the class hierarchy, meaning every class in Java implicitly extends Object. The Object class provides basic methods that are inherited by all Java classes, such as toString(), equals(), and hashCode().

Q. Then can you override the methods in Object?

Ans: Yes, you can override the methods in the Object class. For example, you can override methods like toString(), equals(), and hashCode() in your own classes to provide custom behavior. This is a common practice to ensure that your classes behave appropriately in collections and other frameworks that rely on these methods.

Q. HOW can you let somebody make an Object object? Isn't that just as weird as making an Animal object?

Ans: Creating an instance of Object is generally unnecessary because it doesn't provide any useful functionality beyond what is inherited by other classes. It is similar to creating an Animal object if

Animal were an abstract class with no specific behavior. Instead, you typically create instances of more specific subclasses that inherit from Object, which provide meaningful functionality.

Q. So is it fair to say that the main purpose for type Object is so that you can use it for a polymorphic argument and return type?

Ans: Yes, that's correct. The main purpose of Object is to serve as a base type for all Java classes, enabling polymorphism. It allows you to use it as a parameter type or return type for methods that need to handle different types of objects in a flexible manner.

Q. If it's so good to use polymorphic types, why don't you just make ALL your methods take and return type Object?

Ans: Using Object as a parameter or return type for all methods would lead to a loss of type safety and require extensive type casting, which can introduce errors and reduce code clarity. It's better to use more specific types where possible to take advantage of compile-time type checking, which helps ensure that methods are used correctly and improves code readability and maintainability.

Q. Wait a minute, interfaces don't really give you multiple inheritance, because you can't put any implementation code in them. If all the methods are abstract, what does an interface really buy you?

Ans: Interfaces provide a way to define a contract that classes can implement, ensuring that they adhere to certain methods. They enable a form of multiple inheritance by allowing a class to implement multiple interfaces, thereby inheriting various method signatures without inheriting implementation. This supports flexibility in design and allows classes to implement behaviors from multiple sources, promoting loose coupling and better code organization.

Q. What if you make a concrete subclass and you need to override a method, but you want the behavior in the superclass version of the method? In other words, what if you don't need to replace the method with an override, but you just want to add to it with some additional specific code.

Ans: You can achieve this by calling the superclass's method within your overridden method using super.methodName(). This allows you to retain the superclass's behavior while adding additional functionality specific to the subclass.

Q. There's still something strange here...you never explained how it is that ArrayList gives back Dog references that don't need to be cast. What's the special trick going on when you say ArrayList?

Ans: The trick is the use of generics in Java. When you declare ArrayList<Dog>, you're specifying that the ArrayList will only hold Dog objects. This ensures type safety and eliminates the need for casting when retrieving objects. Generics allow you to enforce type constraints at compile time, so you get type-checking and compiler warnings if you try to add an object of a different type.

WHAT'S THE DECLARATION?(pg 233)

1. public class Click { }
   public class Clack extends Click { }

2. public abstract class Top { }
   public class Tip extends Top { }

3. public abstract class Fee { }
   public abstract class Fi extends Fee { }

4. public interface Foo { }
   public class Bar implements Foo { }

public class Baz extends Bar { }

5. public interface Zeta { }

   public class Alpha implements Zeta { }

   public interface Beta { }

   public class Delta extends Alpha implements Beta { }

CHAPTER 9 – LIFE AND DEATH OF AN OBJECT

THERE ARE NO DUMB QUESTIONS(Pg:240)

Q) One more time, WHY are we learning the whole stack/heap thing? How does this help me? Do I really need to learn about it?

Ans: Learning about stack and heap memory is essential for understanding how Java manages memory, which affects application performance and behavior. Stack memory handles method calls and local variables, helping with debugging method execution and variable scope issues. Heap memory, where objects are allocated, is crucial for optimizing memory usage and managing garbage collection. Understanding these memory areas aids in writing efficient code, avoiding memory leaks, and improving application performance.

SHARPEN YOUR SKILLS(Pg 244)

Q) A constructor lets you jump into the middle of the object creation step—into the middle of new. Can you imagine conditions where that would be useful? Which of the actions on the right might be useful in a Car class constructor, if the Car is part of a Racing Game? Check off the ones that you came up with a scenario for.

Ans:

- Increment a counter to track how many objects of this class type have been made.
- Assign runtime-specific state (data about what's happening NOW).
- Assign values to the object's important instance variables.
- Create HAS-A objects.

THERE ARE NO DUMB QUESTIONS

Pg:245

Q: Why do you need to write a constructor if the compiler writes one for you?

Ans: A custom constructor allows for specific initialization of an object with custom values or setup, which the default constructor provided by the compiler does not do. This ensures the object is properly configured for its intended use.

Q) **How can you tell a constructor from a method? Can you also have a method that's the same name as the class?**

Ans: A constructor can be distinguished from a method by its name and its lack of a return type. A constructor has the same name as the class and does not specify a return type, not even `void`. Methods, on the other hand, have their own names, include a return type, and can be called to perform specific operations. While you cannot have a method with the exact same name as the class, you can have multiple constructors with different parameter lists (constructor overloading), but the method name must differ from the class name.

Q: **Are constructors inherited? If you don't provide a constructor but your superclass does, do you get the superclass constructor instead of the default?**

Ans: Constructors are not inherited. If a subclass does not provide any constructors, it receives a default constructor provided by the compiler. This default constructor will call the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor and the subclass does not provide any constructors, the code will not compile. To use a superclass constructor with parameters, the subclass must explicitly define a constructor and call the appropriate superclass constructor using `super()`.

Q) Earlier you said that it's good to have a no-argument constructor so that if people call the no-arg constructor, we can supply default values for the "missing" arguments. But aren't there times when it's impossible to come up with defaults? Are there times when you should not have a no-arg constructor in your class?

Ans: Yes, there are times when providing a no-argument constructor with default values might not be appropriate. If a class requires essential initialization data that cannot be reasonably defaulted, or if the state of the object is critical and must be explicitly defined, a no-argument constructor might not be suitable. In such cases, constructors with parameters should be used to ensure that objects are created with the necessary information. Providing only parameterized constructors can enforce that all required data is supplied, avoiding potential issues with incomplete or invalid object states.

Pg 251

Q: Do constructors have to be public?

Ans: No. Constructors can be public, protected, private, or default.

Q: How could a private constructor ever be useful? Nobody could ever call it, so nobody could ever make a new object!

Ans: A private constructor can be useful for controlling object creation in specific scenarios. For instance, it is essential in implementing the Singleton pattern, where only one instance of a class should exist, and the private constructor ensures that no other instances can be created directly. It is also used in static utility classes to prevent instantiation, as these classes are meant to contain only static methods. Additionally, private constructors can be employed with factory methods to manage how objects are created and initialized, allowing for controlled and flexible instance management. This approach ensures that object creation adheres to specific rules or constraints defined by the class.

SHARPEN YOUR PENCIL(Pg 254)

Q) What's the real output? Given the code on the left, what prints out when you run TestHippo? A or B?

A:

% java TestHippo

Starting...

Making an Animal

Making a Hippo

B:

java TestHippo

Starting...

Making a Hippo

Making an Animal

Ans: The first one, A. The Hippo() constructor is invoked first, but it's the Animal constructor that finishes first.

EXERCISE(Pg 268)

Q) Which of the lines of code on the right, if added to the class on the left at point A, would cause exactly one additional object to be eligible for the Garbage Collector? (Assume that point A (//call more methods) will execute for a long time, giving the Garbage Collector time to do its stuff.

public class GC {

public static GC doStuff() {

GC newGC = new GC();

```
doStuff2(newGC);

return newGC;

}

public static void main(String[] args) {

GC gc1;

GC gc2 = new GC();

GC gc3 = new GC();

GC gc4 = gc3;

gc1 = doStuff();

// call more methods

}

public static void doStuff2(GC copyGC) {

GC localGC = copyGC;

}

}
```

Ans:

copyGC = null;	→	No—this line attempts to access a variable that is out of scope.
gc2 = null;	→	OK—gc2 was the only reference variable referring to that object.
newGC = gc3;	→	No—another out of scope variable.
gc1 = null;	→	OK—gc1 had the only reference because newGC is out of scope.
newGC = null;	→	No—newGC is out of scope.
gc4 = null;	→	No—gc3 is still referring to that object.
gc3 = gc2;	→	No—gc4 is still referring to that object.

gc1 = gc4; → OK—Reassigning the only reference to that object.

gc3 = null; → No—gc4 is still referring to that object.

Q) In this code example, several new objects are created. Your challenge is to find the object that is "most popular," i.e., the one that has the most reference variables referring to it. Then list how many total references there are for that object, and what they are! We'll start by pointing out one of the new objects and its reference variable.

```
class Bees {

Honey[] beeHoney;

}

class Raccoon {

Kit rk;

Honey rh;

}

class Kit {

Honey honey;

}

class Bear {

Honey hunny;

}

public class Honey {

public static void main(String[] args) {

Honey honeyPot = new Honey();

Honey[] ha = {honeyPot, honeyPot, honeyPot, honeyPot};
```

```
Bees bees = new Bees();

bees.beeHoney = ha;

Bear[] bears = new Bear[5];

for (int i = 0; i < 5; i++) {

bears[i] = new Bear();

bears[i].hunny = honeyPot;

}

Kit kit = new Kit();

kit.honey = honeyPot;

Raccoon raccoon = new Raccoon();

raccoon.rh = honeyPot;

raccoon.rk = kit;

kit = null;

} // end of main
```

Ans:

In the given code, the `Honey` object `honeyPot` is the most popular, having the most reference variables referring to it. Here is a list of all the references to `honeyPot`:

1. `honeyPot` (direct reference in the `main` method).

2. `ha[0]`, `ha[1]`, `ha[2]`, `ha[3]` (array elements in `ha`).

3. `bees.beeHoney[0]`, `bees.beeHoney[1]`, `bees.beeHoney[2]`, `bees.beeHoney[3]` (references in the `Bees` object).

4. `bears[0].hunny`, `bears[1].hunny`, `bears[2].hunny`, `bears[3].hunny`, `bears[4].hunny` (references in each `Bear` object in the `bears` array).

5. `kit.honey` (reference in the `Kit` object, which becomes `raccoon.rk.honey` after `kit` is nullified).

6. `raccoon.rh` (reference in the `Raccoon` object).

Therefore, the total number of references to the `honeyPot` object is 15.

CHAPTER 10 – NUMBERS MATTER

THERE ARE NO DUMB QUESTIONS(Pg 280)

Q: What if you try to call a non-static method from a static method, but the non-static method doesn't use any instance variables. Will the compiler allow that?

Ans: The compiler will not allow calling a non-static method directly from a static method, even if the non-static method does not use any instance variables. Non-static methods are associated with instances of the class and require an object of the class to be called. Static methods belong to the class itself rather than any particular instance. To call a non-static method from a static context, one must first create an instance of the class and then invoke the non-static method on that instance.

Q: I could swear I've seen code that calls a static method using a reference variable instead of the class name.

Ans: It is indeed possible to call a static method using a reference variable, even though it is not the conventional approach. The compiler allows this because static methods belong to the class itself, not to instances. Hence, a static method can be accessed via an instance reference, though it is typically preferred to call it using the class name for clarity and to emphasize that it is a class-level method. Using an instance reference to call a static method does not create a new instance or change the method's behavior but can lead to confusion about the method's static nature.

Pg 286:

Q: A static method can't access a non-static variable. But can a non-static method access a static variable?

Ans: A non-static method in a class can always call a static method in the class or access a static variable of the class.

Q: Why would I want to make a class final? Doesn't that defeat the whole purpose of OO?

Ans: Making a class `final` in Java prevents it from being subclassed, which can be advantageous in several scenarios. It can ensure that the class's implementation remains consistent and cannot be altered through inheritance, thus providing greater control over its behavior and maintaining immutability. This approach is useful for creating immutable objects, such as the `String` class, or for classes that are intended to be utility classes with static methods, like `java.util.Collections`. Declaring a class `final` does not defeat the purpose of object-oriented programming; instead, it helps enforce design constraints and promote reliable and predictable behavior.

Q: Isn't it redundant to have to mark the methods final if the class is final?

Ans: It is not redundant to mark methods as `final` even if the class itself is `final`. Declaring a class as `final` prevents subclassing, so no further inheritance can occur, and therefore, no method overrides are possible. However, marking methods as `final` explicitly communicates the intent that these methods should not be overridden if the class were to be subclassed in other contexts. It also serves as a form of documentation for developers, indicating that the method's behavior is intended to be final and should not be altered, even though the class itself cannot be extended.

SHARPEN YOUR PENCIL(Pg: 287)

Q) Given everything you've just learned about static and final, which of these would compile?

Ans:

1)

```
public class Foo {

static int x;
```

```
public void go() {
System.out.println(x);
}
}
```

Ans: The above code will compile

2)
```
public class Foo2
 { int x;
 public static void go()
{
System.out.println(x);
}
 }
```

Ans: The above code will not compile

3)
```
public class Foo3 {
final int x;
public void go() {
System.out.println(x);
}
}
```

Ans: The above code will compile

4) public class Foo4 {

static final int x = 12;

public void go() {

System.out.println(x);

}

}

Ans: The above code compiles

5) public class Foo5 {

static final int x = 12;

public void go(final int x) {

System.out.println(x);

}

}

Ans: The above code compiles

6) public class Foo6 {

int x = 12;

public static void go(final int x) {

System.out.println(x);

}

}

Ans: The above code does not compile

SHARPEN YOUR PENCILS(pg 293)

Q) Will this code compile? Will it run? If it runs, what will it do?

```
public class TestBox {

private Integer i;

private int j;

public static void main(String[] args) {

TestBox t = new TestBox();

t.go();

}

public void go() {

j = i;

System.out.println(j);

System.out.println(i);

}

}
```

Ans: The code will run but when 'i' which is null is assigned to j, it throws a NullPointerException

THERE ARE NO DUMB QUESTIONS (Pg: 302)

Q) Um, there's something REALLY strange going on here. Just how many arguments *can* I pass? I mean, how many overloaded format() methods are IN the String class? So, what happens if I want to pass, say, ten different arguments to be formatted for a single output String?

Ans: The `String` class in Java provides several overloaded `format()` methods that can handle varying numbers of arguments, up to a maximum of nine arguments directly. For more than nine

arguments, you would need to use an array or a collection to pass the arguments to the `format()` method.

BE THE COMPILER (Pg 306)

Q) The Java file on this page represents a complete program. Your job is to play compiler and determine whether this file will compile. If it won't compile, how would you fix it? When it runs, what would be its output?

```
class StaticSuper {

static {

System.out.println("super static block");

}

StaticSuper () {

System.out.println("super constructor");

}

}

public class StaticTests extends StaticSuper {

static int rand;

static {

rand = (int) (Math.random() * 6);

System.out.println("static block " + rand);

}

StaticTests() {

System.out.println("constructor");

}
```

```
public static void main(String[] args) {

System.out.println("in main");

StaticTests st = new StaticTests();

}

}
```

Ans: The above code will not compile, in order to fix it, we need to make the following change in the code since StaticSuper is a constructor and must have ( ) in its signature.

```
StaticSuper () {

System.out.println(

"super constructor");

}
```

Output:

```
super static block

static block 3

in main

super constructor

constructor
```

TRUE OT FALSE (Pg 307)

Q) True or False

1. To use the Math class, the first step is to make an instance of it- false

2. You can mark a constructor with the keyword “static.” - false

3. Static methods don’t have access to an object’s instance variables- true

4. It is good practice to call a static method using a reference variable- false

5. Static variables could be used to count the instances of a class- true

6. Constructors are called before static variables are initialized- false

7. MAX_SIZE would be a good name for a static final variable- true

8. A static initializer block runs before a class's constructor runs- true

9. If a class is marked final, all of its methods must be marked final- false

10. A final method can be overridden only if its class is extended- false

11. There is no wrapper class for boolean primitives- false

12. A wrapper is used when you want to treat a primitive like an object- true

13. The parseXxx methods always return a String- false

14. Formatting classes (which are decoupled from I/O) are in the java.format package- false

CHAPTER 11 – DATA STRUCTURES

THERE ARE NO DUMB QUESTIONS

Pg 313

Q: Should I use the diamond operator (<>) all the time? Any downsides?

Ans: The diamond operator, also known as type inference for generic types, simplifies code by allowing you to omit redundant type information when creating instances of generic classes. It doesn't affect the underlying bytecode; it's purely syntactic sugar. You can use it confidently—it's equivalent to specifying the type explicitly. However, consider using the full type when initialization is distant from the declaration. Clarity matters for readability.

Q: Are there other places where the compiler can infer types for me?

Ans: Yes. The var keyword (Local Variable Type Inference) allows the compiler to infer the type of local variables based on their initialization. Additionally, lambda expressions (covered later) benefit from type inference, making code more concise and expressive.

Q: Why not use ArrayList everywhere instead of assigning it to a List reference?

Ans: Polymorphism plays a role here. Code doesn't need to know the specific implementation type (e.g., ArrayList) to work with objects. By using the interface type (List), you ensure flexibility. Other code interacting with the list only needs to rely on common methods (e.g., add(), size()). This practice allows seamless transitions—switching from ArrayList to other list implementations (e.g., LinkedList, CopyOnWriteArrayList) without altering dependent code.

Pg 322

Q: But don't I also need to learn how to create my OWN generic classes? What if I want to make a class type that lets people instantiating the class decide the type of things that class will use?

Ans: While it's true that you might encounter scenarios where creating your own generic classes is necessary, in practice, most of your work will involve using existing libraries. The Java API already provides a comprehensive set of generic collection classes, covering various data structures. These built-in classes handle most of your needs.

Pg 323

Q: Is "E" the only thing you can put there? Because the docs for sort used "T"....?

Ans: No, you're not limited to just "E." In Java generics, you can use any legal Java identifier as a type parameter. Typically, single letters like "T" (for "Type") are common. However, when specifically writing collection classes, it's customary to use "E" to represent the type of elements the collection will hold. Additionally, you might encounter "R" for "Return type" in certain contexts.

Pg 328

Q. Why didn't they just make a new keyword, "is"?

Ans: Adding new keywords to the language is a significant decision because it risks breaking existing Java code. For example, if we use a variable named "is"; and if "is" became a reserved keyword, any earlier code using it would break. To avoid this, language engineers often reuse existing keywords (like "extends") whenever possible. However, in recent years, Java has introduced "keyword-like" terms (e.g., var) that don't break existing code.

Q. Why are there two sort methods that take a comparator on two different classes? Which sort method should I use?

Ans. Both methods that take a comparator, Collections.sort(List, Compara tor) and List.sort(Comparator), perform the same function; either can be used for the same results. List.sort was introduced in Java 8, so older code must use Collections. sort(List, Comparator). We use List.sort because it's a bit shorter, and generally we already have the list you want to sort, so it makes sense to call the sort method on that list.

Pg 351

Q. How come hash codes can be the same even if objects aren't equal?

Ans: HashSets use hash codes to organize elements efficiently. When you want to find an object in a HashSet, it quickly locates the corresponding "bucket" based on the hash code. This approach is faster than searching through an ArrayList sequentially. While hash codes can be the same for different objects, the HashSet handles collisions by using the equals() method to ensure correctness.

Q. Why are there two sort methods that take a comparator on two different classes? Which sort method should I use?

Ans. Both methods that take a comparator, Collections.sort(List, Compara tor) and List.sort(Comparator), perform the same function; either can be used for the same results. List.sort was introduced in Java 8, so older code must use Collections. sort(List, Comparator). We use List.sort because it's a bit shorter, and generally we already have the list you want to sort, so it makes sense to call the sort method on that list.

SHARPEN YOUR PENCIL

Pg 329

Q. Write in your idea and pseudocode (or better, REAL code) for implementing the compareTo() method in a way that will sort() the Song objects by title. Hint: if you're on the right track, it should take less than three lines of code!

Ans:

```
public int compareTo(SongV3 s) {
   return title.compareTo(s.getTitle());
 }
```

Q. Fill-in-the-blanks for each of the questions below

Given the following compilable statement:

Collections.sort(myArrayList);

1. What must the class of the objects stored in myArrayList implement?

   Comparable

2. What method must the class of the objects stored in myArrayList implement?

   compareTo()

3. Can the class of the objects stored in myArrayList implement both Comparator AND Comparable?

   yes

Given the following compilable statements (they both do the same thing):

Collections.sort(myArrayList, myCompare);

myArrayList.sort(myCompare);

4. Can the class of the objects stored in myArrayList implement Comparable?

yes

5. Can the class of the objects stored in myArrayList implement Comparator?

yes

6. Must the class of the objects stored in myArrayList implement Comparable?

no

7. Must the class of the objects stored in myArrayList implement Comparator?

no

8. What must the class of the myCompare object implement?

Comparator

9. What method must the class of the myCompare object implement?

Compare()

Pg 342

Q. Write lambda expressions to sort the songs in these ways:

- Sort by BPM
- Sort by title in descending order

Ans:

- songList.sort((one, two) -> one.getBpm() - two.getBpm());
- songList.sort((one, two) -> two.getTitle().compareTo(one.getTitle()));

Pg 343

Q. Reverse Engineer

Assume this code exists in a single file. Your job is to fill in the blanks so the program will create the output shown

```
import java.util.*;

public class SortMountains {

 public static void main(String[] args) {

   new SortMountains().go();

 }

 public void go() {

   List<Mountain> mountains = new ArrayList<>();

   mountains.add(new Mountain("Longs", 14255));

   mountains.add(new Mountain("Elbert", 14433));

   mountains.add(new Mountain("Maroon", 14156));

   mountains.add(new Mountain("Castle", 14265));

   System.out.println("as entered:\n" + mountains);

   mountains.sort((mount1, mount2) -> mount1.name.compareTo(mount2.name));

   System.out.println("by name:\n" + mountains);

   mountains.sort((mount1, mount2) -> mount2.height - mount1.height);

   System.out.println("by height:\n" + mountains);

 }

}

 class Mountain {
```

```
String name;
int height;
Mountain(String name, int height) {
this.name = name;
this.height = height;
 }
public String toString( ) {
return name + " " + height;
 }
}
```

Pg 353

Q. Look at this code. Read it carefully, then answer the questions below.

```
import java.util.*;
public class TestTree {
 public static void main(String[] args) {
   new TestTree().go();
 }
 public void go() {
   Book b1 = new Book("How Cats Work");
   Book b2 = new Book("Remix your Body");
   Book b3 = new Book("Finding Emo");
   Set<Book> tree = new TreeSet<>();
```

```
  tree.add(b1);
  tree.add(b2);
  tree.add(b3);
  System.out.println(tree);
 }
}
class Book {
 private String title;
 public Book(String t) {
  title = t;
 }
}
```

1. What is the result when you compile this code?

   It compiles correctly

2. If it compiles, what is the result when you run the TestTree class?

   It throws an exception:

   Exception in thread "main" java.lang.ClassCastException: class Book cannot be cast to class java.lang.Comparable

   at java.base/java.util.TreeMap.compare(TreeMap.java:1291)

   at java.base/java.util.TreeMap.put(TreeMap.java:536)

   at java.base/java.util.TreeSet.add(TreeSet.java:255)

   at TestTree.go(TestTree.java:16)

at TestTree.main(TestTree.java:7)

3. If there is a problem (either compile-time or runtime) with this code, how would you fix it?

Make Book implement Comparable, or pass the TreeSet a Comparator

BE THE COMPILER (Pg 363):

Q. Compiles?

- ✓ takeAnimals(new ArrayList<Animal>());
- × takeDogs(new ArrayList<Animal>());
- × takeAnimals(new ArrayList<Dog>());
- ✓ takeDogs(new ArrayList<>());
- ✓ List<Dog> dogs = new ArrayList<>();
  takeDogs(dogs);
- ✓ takeSomeAnimals(new ArrayList<Dog>());
- ✓ takeSomeAnimals(new ArrayList<>());
- ✓ takeSomeAnimals(new ArrayList<Animal>());
- ✓ List<Animal> animals = new ArrayList<>();
  takeSomeAnimals(animals);
- × List<Object> objects = new ArrayList<>();
  takeObjects(objects);
- × takeObjects(new ArrayList<Dog>());
- ✓ takeObjects(new ArrayList<Object>());

CHAPTER 12 – LAMBDAS AND STREAMS: WHAT, NOT HOW

BE THE COMPILER (Pg 363):

Q. Compiles?

- ✓ Runnable r = () -> System.out.println("Hi!");
- ✓ Consumer <String> c = s -> System.out.println(s);
- × Supplier <String> s = () -> System.out.println("Some string");
- × Consumer <String> c = (s1, s2) -> System.out.println(s1 + s2);
- × Runnable r = (String str) -> System.out.println(str);
- ✓ Function f <String, Integer>= s -> s.length();
- ✓ Supplier s <String> = () -> "Some string";
- × Consumer <String> c = s -> "String" + s;
- × Function<String, Integer> f = (int i) -> "i = " + i;
- × Supplier <String> s = s -> "Some string: " + s;
- × Function<String> f = () -> System.out.println("Some string");

CODE MAGNETS (Pg 386)

Q. A Java program is all scrambled up on the fridge. Can you reconstruct the code snippets to make a working Java program that produces the output listed below?

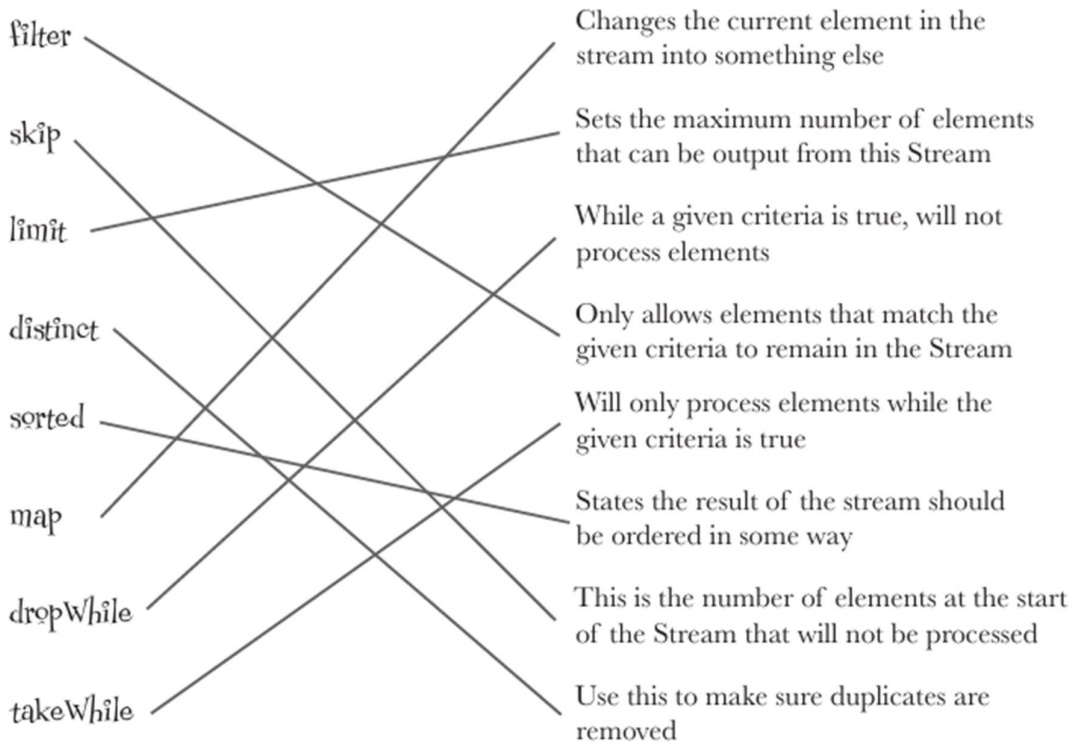
Ans.

```
import java.util.*;
import java.util.stream.*;
public class CoffeeOrder {
 public static void main(String[] args) {
```

```
    List<String> coffees = List.of("Cappuccino", "Americano", "Espresso", “Cortado", "Mocha",
"Cappuccino", "Flat White", "Latte");
    List<String> coffeesEndingInO = coffees.stream()
                                    .filter(s -> s.endsWith("o"))
                                    .sorted()
                                    .distinct()
                                    .collect(Collectors.toList());
    System.out.println(coffeesEndingInO);
  }
}
```

WHO DOES WHAT? (Pg 374)

Q. Have a go at matching each operation name to the description of what it does:



THERE ARE NO DUMB QUESTIONS

Pg 387

Q: Is there any point in having a stream pipeline without intermediate operations?

Ans: Yes, there are scenarios where a terminal operation alone suffices to transform the original collection into a desired shape. However, be aware that some terminal operations (like count) have equivalents available directly on the collection (e.g., size for Lists). Additionally, if your collection is already iterable (e.g., a List), you can use the existing forEach method without resorting to streams.

Q: How can I avoid changing the source collection during a stream operation?

Ans: While it's possible to write concurrent programs where different code runs simultaneously, it's generally safer to create collections that cannot be modified if they don't need to change. This principle applies not only to Streams but also to other contexts.

Q: How can I obtain an unmodifiable List from the collect terminal operation?

Ans: If you're using Java 10 or higher, utilize Collectors.toUnmodifiableList instead of Collectors.toList when calling collect.

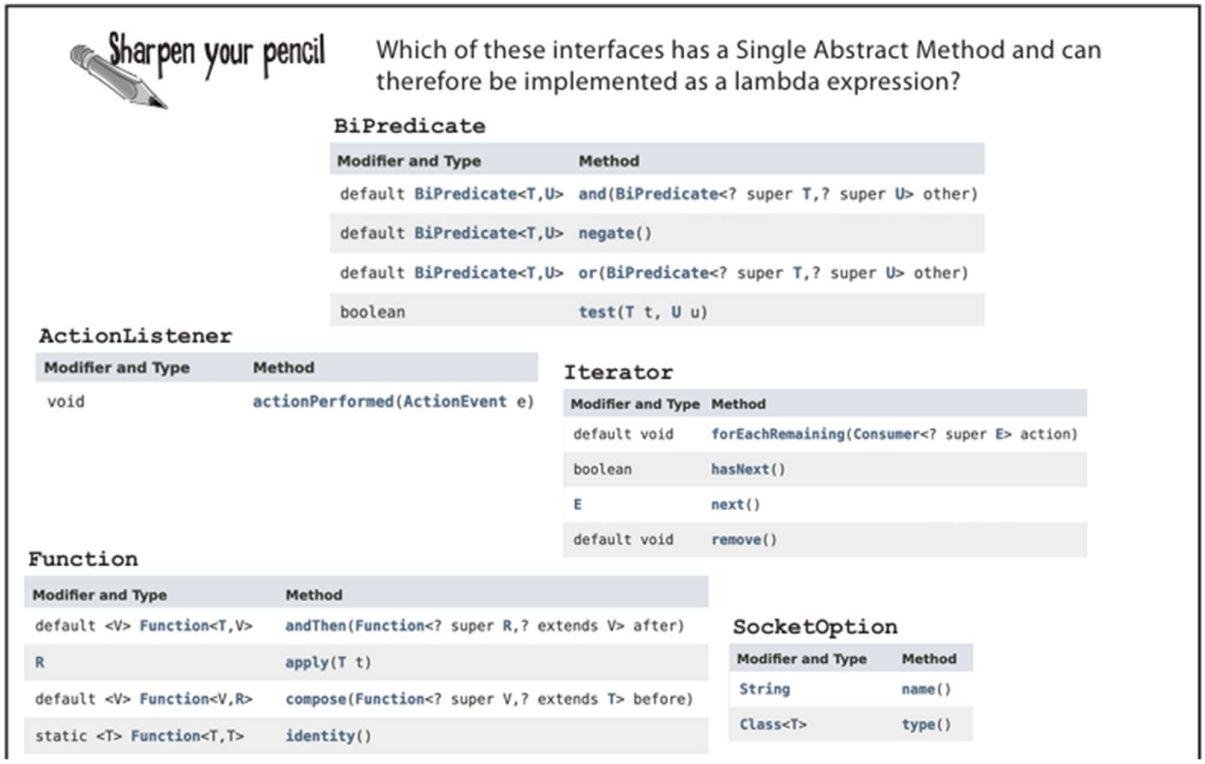
Q: Can I collect stream results into a collection other than a List?

Ans: Yes. The Collectors class provides convenience methods for collecting into various collection types. You can use toSet, toMap, and (since Java 10) toUnmodifiableSet and toUnmodifiableMap.

SHARPEN YOUR PENCIL

Pg 397

Q. Which of these interfaces has a Single Abstract Method and can therefore be implemented as a lambda expression?



Ans: Interfaces BiPredicate, ActionListener, and Function all have a Single Abstract Method and can therefore be implemented as a lambda expression.

READY TO BAKE CODE EXERCISE:

Pg 398

Updated Code:

```
import java.util.List;

class Songs {

   public List<Song> getSongs() {

      return List.of(
```

```
            new Song("$10", "Hitchhiker", "Electronic", 2016, 183),
            // ... (other songs)
            new Song("Smooth", "Santana", "Latin", 1999, 244),
            new Song("Immigrant song", "Led Zeppelin", "Rock", 1970, 484)
        );
    }
    public static void main(String[] args) {
        Songs jukebox = new Songs();
        List<Song> allSongs = jukebox.getSongs();
        for (Song song : allSongs) {
            System.out.println(song);
        }
    }
}
class Song {
    private final String title;
    private final String artist;
    private final String genre;
    private final int year;
    private final int timesPlayed;

    public Song(String title, String artist, String genre, int year, int timesPlayed) {
        this.title = title;
```

```
        this.artist = artist;
        this.genre = genre;
        this.year = year;
        this.timesPlayed = timesPlayed;
    }
    public String toString() {
        return title + " by " + artist + " (" + year + ")";
    }
}
```

Expected Output:

$10 by Hitchhiker (2016)

Havana by Camila Cabello (2017)

Cassidy by Grateful Dead (1972)

50 ways by Paul Simon (1975)

Hurt by Nine Inch Nails (1995)

Silence by Delerium (1999)

Hurt by Johnny Cash (2002)

Watercolour by Pendulum (2010)

The Outsider by A Perfect Circle (2004)

With a Little Help from My Friends by The Beatles (1967)

Come Together by The Beatles (1968)

Come Together by Ike & Tina Turner (1970)

With a Little Help from My Friends by Joe Cocker (1968)

Immigrant Song by Karen O (2011)

Breathe by The Prodigy (1996)

What's Going On by Gaye (1971)

Hallucinate by Dua Lipa (2020)

Walk Me Home by P!nk (2019)

I am not a woman, I'm a god by Halsey (2021)

Pasos de cero by Pablo Alborán (2014)

Smooth by Santana (1999)

Immigrant song by Led Zeppelin (1970)

CHAPTER 13 – RISKY BEHAVIOR

THERE ARE NO DUMB QUESTIONS (Pg 430)

Q: Wait just a minute! How come this is the FIRST time we've had to try/catch an Exception? What about the exceptions I've already gotten like NullPointerException and the exception for DivideByZero? I even got a NumberFormatException from the Integer.parseInt() method. How come we didn't have to catch those?

Ans: The compiler cares about all subclasses of Exception, except for RuntimeException. Any class extending RuntimeException doesn't require explicit handling (throws declaration or try/catch blocks). These exceptions can occur anywhere during runtime.

Q: I'll bite. WHY doesn't the compiler care about those runtime exceptions? Aren't they just as likely to bring the whole show to a stop?

Ans: Most RuntimeExceptions stem from code logic issues rather than unpredictable conditions. While you can't guarantee external factors (like file existence or server availability), you can prevent array index errors (using .length). Detect RuntimeExceptions during development and testing—they shouldn't occur in production. Use try/catch for exceptional situations, not code flaws.

Q: How can I obtain an unmodifiable List from the collect terminal operation?

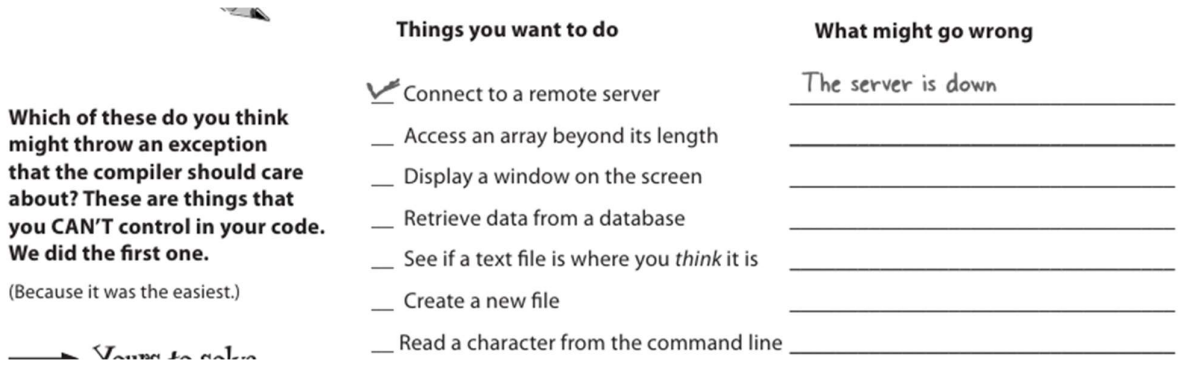
Ans: If you're using Java 10 or higher, utilize Collectors.toUnmodifiableList instead of Collectors.toList when calling collect.

Q: Can I collect stream results into a collection other than a List?

Ans: Yes. The Collectors class provides convenience methods for collecting into various collection types. You can use toSet, toMap, and (since Java 10) toUnmodifiableSet and toUnmodifiableMap.

SHARPEN YOUR PENCIL

Pg 431



Which of these do you think might throw an exception that the compiler should care about? These are things that you CAN'T control in your code. We did the first one.

(Because it was the easiest.)

Things you want to do	What might go wrong
✓ Connect to a remote server	The server is down
__ Access an array beyond its length	
__ Display a window on the screen	
__ Retrieve data from a database	
__ See if a text file is where you *think* it is	
__ Create a new file	
__ Read a character from the command line	

Things you want to do	What might go wrong
Connect to a remote server	The server is down, network issues, incorrect server address, authentication failure
Access an array beyond its length	Index out of bounds exception
Display a window on the screen	Graphics system failure, insufficient resources, invalid window parameters
Retrieve data from a database	Database connection failure, SQL errors, incorrect query, data not found
See if a text file is where you think it is	File not found, incorrect file path, permissions issues
Create a new file	Permissions issues, disk full, invalid file name
Read a character from the command line	Input/output error, end of input stream, invalid input format

Pg 434

Q. Look at the code to the left. What do you think the output of this program would be? What do you think it would be if the third line of the program were changed to String test = "yes";? Assume ScaryException extends Exception.

```
public class TestExceptions {
  public static void main(String[] args) {
    String test = "no";
    try {
      System.out.println("start try");
      doRisky(test);
      System.out.println("end try");
    } catch (ScaryException se) {
      System.out.println("scary exception");
    } finally {
      System.out.println("finally");
    }
    System.out.println("end of main");
  }
  static void doRisky(String test) throws ScaryException {
    System.out.println("start risky");
    if ("yes".equals(test)) {
      throw new ScaryException();
    }
    System.out.println("end risky");
  } }
```

```
class ScaryException extends Exception {

}
```

Ans:

Output when test = "no"

start try

start risky

end risky

end try

finally

end of main

Output when test = "yes"

start try

start risky

scary exception

finally

end of main

CODE MAGNETS (Pg 455)

Q. A working Java program is scrambled up on the fridge. Can you reconstruct all the code snippets to make a working Java program that produces the output listed below? Some of the curly braces fell on the floor and they were too small to pick up, so feel free to add as many of those as you need!

Ans:

```
class MyEx extends Exception { }

 public class ExTestDrive {

  public static void main(String[] args) {

    String test = args[0];

    try {

      System.out.print("t");

      doRisky(test);

      System.out.print("o");

    } catch (MyEx e) {

      System.out.print("a");

    } finally {

      System.out.print("w");

    }

    System.out.println("s");

  }

  static void doRisky(String t) throws MyEx {

    System.out.print("h");

    if ("yes".equals(t)) {

      throw new MyEx();

    }

    System.out.print("r");

  }

}
```

EXERCISE: TRUE OR FALSE (Pg 455)

exercise: True or False

This chapter explored the wonderful world of exceptions. Your job is to decide whether each of the following exception-related statements is true or false.

TRUE OR FALSE

1. A try block must be followed by a catch and a finally block.

2. If you write a method that might cause a compiler-checked exception, you must wrap that risky code in a try/catch block.

3. Catch blocks can be polymorphic.

4. Only "compiler checked" exceptions can be caught.

5. If you define a try/catch block, a matching finally block is optional.

6. If you define a try block, you can pair it with a matching catch or finally block, or both.

7. If you write a method that declares that it can throw a compiler-checked exception, you must also wrap the exception throwing code in a try/catch block.

8. The main( ) method in your program must handle all unhandled exceptions thrown to it.

9. A single try block can have many different catch blocks.

10. A method can throw only one kind of exception.

11. A finally block will run regardless of whether an exception is thrown.

12. A finally block can exist without a try block.

13. A try block can exist by itself, without a catch block or a finally block.

14. Handling an exception is sometimes referred to as "ducking."

15. The order of catch blocks never matters.

16. A method with a try block and a finally block can optionally declare a checked exception.

17. Runtime exceptions must be handled or declared.

Ans: True or False

1. False
2. False
3. True.
4. False

5. True.

6. True

7. False

8. False

9. True.

10. False.

11. True

12. False.

13. False.

14. False

15. False.

16. False.

17. False.

CHAPTER 14 – A VERY GRAPHIC STORY

THERE ARE NO DUMB QUESTIONS

Pg 464

Q: I heard that nobody uses Swing anymore.

Ans: While other options like JavaFX exist, there's no definitive answer to which is best for creating GUIs in Java. Learning Swing is still valuable and can help with understanding other frameworks, including Android development.

Q: Will a button look like a Windows button when you run on Windows?

Ans: It can, if you choose. Java provides different "look and feel" options that you can use to control the appearance of your interface, including system-specific styles like the Windows look or cross-platform styles like Metal.

Pg 470

Q: I don’t see the importance of the event object that’s passed to the event call-back methods. If somebody calls my mousePressed method, what other info would I need?

Ans: The event object provides additional information about the event. For instance, in a mousePressed() method, it can tell you the exact X and Y coordinates where the mouse was pressed. This is useful if you want more details than just knowing an event occurred. Additionally, if you register a single listener for multiple components, the event object can help identify which component triggered the event.

Q: Why can’t I be a source of events?

A: You can be an event source. While it's common to start as an event listener, you can create custom events in your applications. For example, if you have a stock market watcher app, you could create a StockMarketEvent and have a StockWatcher object as the event source. You would create a listener interface, like StockListener, with a method such as stockChanged(). When an event occurs, you send it to registered listeners using this method.

SHARPEN YOUR PENCIL

Pg 470

Q. Each of these widgets (user interface objects) is the source of one or more events. Match the widgets with the events they might cause. Some widgets might be a source of more than one event, and some events can be generated by more than one widget.

Ans:
1. Check box - itemStateChanged()
2. Text field - actionPerformed()
3. Scrolling list - itemStateChanged()
4. Button - actionPerformed()
5. Dialog box - windowClosing()
6. Radio button - itemStateChanged()
7. Menu item - actionPerformed()

Pg 493

Q. See if you can design a simple solution to get the ball to animate from the top left of the drawing panel down to the bottom right. Our answer is on the next page, so don’t turn this page until you’re done! Big Huge Hint: make the drawing panel an inner class. Another Hint:

don't put any kind of repeat loop in the paintComponent() method. Write your ideas (or the code) here:

Ans:

```
import javax.swing.*;
 import java.awt.*;
 import java.util.concurrent.TimeUnit;
 public class SimpleAnimation {
  private int xPos = 70;
  private int yPos = 70;
  public static void main(String[] args) {
    SimpleAnimation gui = new SimpleAnimation();
    gui.go();
  }
  public void go() {
    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    MyDrawPanel drawPanel = new MyDrawPanel();
    frame.getContentPane().add(drawPanel);
    frame.setSize(300, 300);
    frame.setVisible(true);
    for (int i = 0; i < 130; i++) {
      xPos++;
      yPos++;
```

```
      drawPanel.repaint();
    }
      try {
        TimeUnit.MILLISECONDS.sleep(50);
      } catch (Exception e) {
        e.printStackTrace();
      }
  }
  class MyDrawPanel extends JPanel {
    public void paintComponent(Graphics g) {
      g.setColor(Color.green);
      g.fillOval(xPos, yPos, 40, 40);
    }
  }
}
```

WHO AM I? (Pg 504)

I got the whole GUI, in my hands.	JFrame
Every event type has one of these.	listener interface
The listener's key method.	actionPerformed( )
This method gives JFrame its size.	setSize( )
You add code to this method but never call it.	paintComponent( )
When the user actually does something, it's an ____ .	event

Most of these are event sources.	swing components
I carry data back to the listener.	event object
An addXxxListener( ) method says an object is an ___ .	event source
How a listener signs up.	addXxxListener( )
The method where all the graphics code goes.	paintComponent( )
I'm typically bound to an instance.	inner class
The "g" in (Graphics g) is really of this class.	Graphics2D
The method that gets paintComponent( ) rolling.	repaint( )
The package where most of the Swingers reside.	javax.swing

BE the COMPILER (Pg 505)

```
import javax.swing.*;
 import java.awt.*;
 import java.awt.event.*;
 class InnerButton {
  private JButton button;
  public static void main(String[] args) {
    InnerButton gui = new InnerButton();
    gui.go();
  }
  public void go() {
    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(
```

```
        JFrame.EXIT_ON_CLOSE);
  button = new JButton("A");
  button.addActionListener(new ButtonListener());
  frame.getContentPane().add(
        BorderLayout.SOUTH, button);
  frame.setSize(200, 100);
  frame.setVisible(true);
 }
 class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
   if (button.getText().equals("A")) {
    button.setText("B");
   } else {
    button.setText("A");
   }
  }
 }
}
```

Note:

The addActionListener( ) method takes a class that implements the ActionListener interface. ActionListener is an interface; interfaces are implemented, not extended. Once this code is fixed, it will create a GUI with a button that toggles between A and B when you click it.

CHAPTER 15 – WORK ON YOUR SWING

THERE ARE NO DUMB QUESTIONS

Pg 522

Q. How come you can't add directly to a frame the way you can to a panel?

Ans: A JFrame interacts with the underlying operating system to display the content, unlike other Swing components, which are pure Java. The content pane is a pure Java layer that sits on top of the JFrame. You can think of the JFrame as the window frame and the content pane as the glass. You can swap the content pane with your own JPanel using myFrame.setContentPane(myPanel); .

Q. Can I change the layout manager of the frame? What if I want the frame to use flow instead of border?

Ans: Yes, you can. The simplest way is to create a panel, set up your GUI in that panel, and then use myFrame.setContentPane(myPanel); to make your panel the frame's content pane, rather than altering the default content pane.

Q. What if I want a different preferred size? Is there a setSize() method for components?

Ans: While there is a setSize() method, layout managers usually ignore it. The preferred size is determined by the component based on its own needs, accessed via getPreferredSize(). The layout manager relies on this method, not setSize().

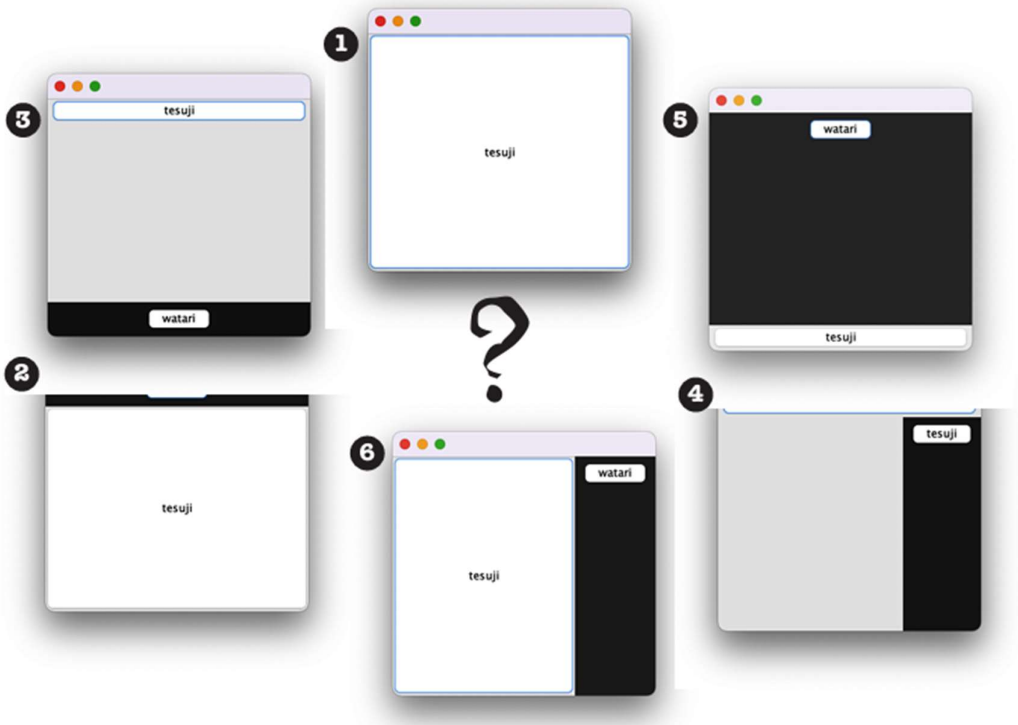
Q. Can't I just put things where I want them? Can I turn the layout managers off?

Ans: Yes, you can call setLayout(null) to disable layout managers and manually set the position and size of components. However, using layout managers is generally easier and more flexible in the long run.

EXERCISE: WHICH LAYOUT (Pg 534)

Which code goes with which layout?

Five of the six screens below were made from one of the code fragments on the opposite page. Match each of the five code fragments with the layout that fragment would produce.



Code Fragments

A

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH, buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

B

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

C

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
```

D

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
```

E

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH, button);
```

Ans:

1 – C

2 – D

3 – E

4 – A

6 - B

CHAPTER 16 – SAVING OBJECTS (AND TEXT)

THERE ARE NO DUMB QUESTIONS(Pg: 550)

Q: **If serialization is so important, why isn't it the default for all classes? Why doesn't class Object implement Serializable, and then all subclasses will be automatically Serializable?**

Ans: Serialization is not the default for all classes in Java because not all objects need to be serialized, and making all objects serializable could lead to unnecessary overhead and security risks. By not implementing `Serializable` in the `Object` class, Java allows developers to have explicit control over which classes should be serializable, ensuring that only the necessary objects are subject to serialization processes. This design decision helps in maintaining optimal performance, memory usage, and security by avoiding unintended serialization of sensitive or non-serializable objects.

Q: **Why would I ever write a class that *wasn't* serializable?**

Ans: The purpose of a class that is serializable is to allow its objects to be converted into a byte stream, which can then be easily saved to a file, transmitted over a network, or stored in a database. This process of serialization enables the persistent storage and transfer of the object's state, allowing it to be reconstructed later through deserialization. This is particularly useful for scenarios such as caching, deep copying, or distributing objects across different systems and components.

Q: **If a class I'm using isn't serializable but there's no good reason, can I subclass the "bad" class and make the *subclass* serializable?**

Ans: Yes, we can create a subclass of the "bad" class and make the subclass serializable by implementing the `Serializable` interface in the subclass. However, we should ensure that the

superclass has a no-argument constructor because during deserialization, the no-argument constructor of the superclass will be called. If the superclass contains fields that need to be serialized, we may need to manually handle their serialization and deserialization in your subclass by overriding the `writeObject` and `readObject` methods.

Q: **You brought it up: what *does* it mean to have a serializable subclass of a non-serializable superclass?**

Ans: Having a serializable subclass of a non-serializable superclass means that while the subclass itself implements `Serializable` and thus can be serialized, the superclass does not. In this case, during serialization, only the state of the subclass that is serializable will be saved. The superclass's state cannot be serialized directly because it does not implement `Serializable`.

When deserializing, the non-serializable superclass's state is not restored, which can lead to incomplete or incorrect object reconstruction. Therefore, if a superclass has fields that are crucial for the integrity of the object, those fields must be either transient (to avoid serialization) or handled in a way that ensures they are properly initialized or restored during the deserialization process.

Q: Whoa! I just realized something big...if you make a variable "transient," this means the variable's value is skipped over during serialization. Then what happens to it? We solve the problem of having a non-serializable instance variable by making the instance variable transient, but don't we NEED that variable when the object is brought back to life? In other words,isn't the whole point of serialization to preserve an object's state?

Ans: When a variable is marked as `transient`, it means that the variable's value is not saved during serialization, and thus, it will not be restored when the object is deserialized. To address this, we must handle the restoration of the transient variable manually. Typically, this is done by initializing the transient variable with a default value or by using custom methods to restore its state after deserialization. For example, you can use the `readObject` method to reinitialize or reconstruct the transient field based on other data or logic. The goal of serialization is indeed to preserve the state of an object, but in cases where certain fields are non-serializable or not meant

to be serialized, we need to ensure that these fields are properly managed during the object's lifecycle, either through initialization or through supplemental data that can be used to restore their state after deserialization.

Q: What happens if two objects in the object graph are the same object? Like, if you have two different Cat objects in the Kennel, but both Cats have a reference to the same Owner object. Does the Owner get saved twice? I'm hoping not.

Ans: No, the `Owner` object will not be saved twice if it is referenced by multiple `Cat` objects during serialization. Java's serialization mechanism handles object graphs and ensures that shared objects are not duplicated in the serialized form. When an object is serialized, its identity is tracked, and if the same object appears more than once in the object graph, it will only be written once. Instead of saving multiple copies of the same object, serialization uses object references to ensure that the same instance is referenced correctly when the object is deserialized. This helps maintain the consistency of object references and prevents redundant storage of identical objects.

Pg 553:

Q) Why doesn't the class get saved as part of the object? That way you don't have the problem with whether the class can be found.

Ans: The class itself is not saved as part of the object during serialization because Java's serialization mechanism relies on the class being available in the classpath at runtime. Saving the class definition with each serialized object would be inefficient and could lead to unnecessary overhead. Instead, serialization includes metadata about the class, such as its name and serial version UID, which helps the deserialization process locate and load the class. The deserialization process expects that the class definition is available in the classpath at runtime, allowing it to reconstruct the object correctly. This design separates the class definition from the serialized object data, keeping the serialized data more compact and manageable while relying on the classpath for class resolution.

Q: What about static variables? Are they serialized?

Ans: Static variables are not serialized. Serialization in Java deals with instance variables, which represent the state of individual object instances. Static variables belong to the class itself rather than to any specific object instance, so they are not part of the object's serialized state. When an object is deserialized, its static variables are initialized based on the class definition as it is loaded at that time, not from the serialized data. This means that the state of static variables is not preserved across serialization and deserialization and is instead determined by the current class definition and any initialization code in the class.

Pg 571

Q: OK, I look in the API and there are about five million classes in the java.io package. How the heck do you know which ones to use?

Ans: The I/O API uses the modular “chaining” concept so that you can join together connection streams and chain streams in a wide range of combinations to get what we want. The chains don’t have to stop at two levels. Multiple chain streams can be combined to one another to get just the right amount of processing we need. If we’re writing text files, we use BufferedReader and BufferedWriter. For serialized objects, ObjectOutputStream and ObjectInputStream can be used.

Pg 576

Q) Wait, what? You told us that a try statement needs a catch and/or a finally?

Ans: In Java, a `try` statement must be followed by at least one `catch` block or a `finally` block if it is used. We can have a `try` block with only a `catch` block to handle exceptions, or just a `finally` block to ensure certain code runs regardless of whether an exception is thrown. Alternatively, you can use both `catch` and `finally` blocks together to handle exceptions and also execute cleanup code. Each configuration serves a different purpose, allowing for flexible exception handling and resource management.

SHARPEN YOUR PENCIL(pg:580)

Q) This version has a huge limitation! When you hit the "serializeIt" button, it serializes automatically, to a file named "Checkbox.ser" (which gets created if it doesn't exist). But each time you save, you overwrite the previously saved file. Improve the save and restore feature by incorporating a JFileChooser so that you can name and save as many different patterns as you like, and load/restore from *any* of your previously saved pattern files.

Ans: To improve the save and restore feature, incorporate a `JFileChooser` to enable users to select file names and locations for saving and loading serialized data. When the "serializeIt" button is clicked, prompt the user with a `JFileChooser` dialog to choose a file name and location, allowing the object to be serialized into the selected file. Similarly, for restoring, use `JFileChooser` to let the user select a previously saved file, and deserialize the object from that file. This approach ensures that files are not overwritten and provides flexibility in managing multiple saved patterns.

Q) Which of these do you think are, or should be, serializable? If not, why not? Not meaningful? Security risk? Only works for the current execution of the JVM? Make your best guess, without looking it up in the API.

Ans:

Object- No, not meaningful

String- Yes

File- No, Security risk.

Date- Yes

OutputStream- No

JFrame- No, Security risk.

Integer- Yes

System- No, not meaningful

Q) What's Legal in the given code fragments

Ans-

Illegal:

```
FileReader fileReader = new FileReader();

BufferedReader reader = new BufferedReader(fileReader);
```

Legal:

```
FileOutputStream f = new FileOutputStream("Foo.ser");

ObjectOutputStream os = new ObjectOutputStream(f);
```

Legal:

```
BufferedReader reader = new BufferedReader(new FileReader(file));

String line;

while ((line = reader.readLine()) != null) {

makeCard(line);

}
```

Illegal:

```
FileOutputStream f = new FileOutputStream("Game.ser");

ObjectInputStream is = new ObjectInputStream(f);

GameCharacter oneAgain = (GameCharacter) is.readObject();
```

EXERCISE:(pg 582)

Q) True or false:

1. Serialization is appropriate when saving data for non-Java programs to use- False

2. Object state can be saved only by using serialization- False

3. ObjectOutputStream is a class used to save serializable objects- True

4. Chain streams can be used on their own or with connection streams- True

5. A single call to writeObject() can cause many objects to be saved- True

6. All classes are serializable by default- False

7. The java.nio.file.Path class can be used to locate files- True

8. If a superclass is not serializable, then the subclass can't be serializable- False

9. Only classes that implement AutoCloseable can be used in try-with-resources statements- False

10. When an object is deserialized, its constructor does not run- True

11. Both serialization and saving to a text file can throw exceptions- True

12. BufferedWriters can be chained to FileWriters- True

13. File objects represent files, but not directories- False

14. You can't force a buffer to send its data before it's full- False

15. Both file readers and file writers can optionally be buffered- True

16. The methods on the Files class let you operate on files and directories- True

17. Try-with-resources statements cannot include explicit finally blocks- False

CODE MAGNETS: (Pg: 583)

Q) Organize the code snippets

Ans:

```
import java.io.*;

class DungeonGame implements Serializable {

public int x = 3;

transient long y = 4;
```

```
private short z = 5;

int getX() {

return x;

}

long getY() {

return y;

}

short getZ() {

return z;

}

}

class DungeonTest {

public static void main(String[] args) {

DungeonGame d = new DungeonGame();

System.out.println(d.getX() + d.getY() + d.getZ());

try {

FileOutputStream fos = new FileOutputStream("dg.ser");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(d);

oos.close();

FileInputStream fis = new FileInputStream("dg.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

d = (DungeonGame) ois.readObject();
```

```
ois.close();

} catch (Exception e) {

e.printStackTrace();

}

System.out.println(d.getX() + d.getY() + d.getZ());

}

}
```

CHAPTER 17 – MAKE A CONNECTION

THERE ARE NO DUMB QUESTIONS(Pg:593)

Q: How do you know the port number of the server program you want to talk to?

Ans: To know the port number of the server program you want to connect to, we need to obtain this information from the server administrator or documentation. The port number is often specified as part of the server's configuration settings. In some cases, it may also be included in the application's documentation or setup instructions. If we are developing both the client and server applications, you would specify and use the port number within the server's code or configuration, and then use the same port number in the client code to establish a connection.

Q) Can there ever be more than one program running on a single port? In other words, can two applications on the same server have the same port number?

Ans: It is not possible for more than one program to run on the same port number on a single IP address. Ports are unique endpoints for communication within an IP address, and each port number can only be used by one application at a time. If a second application attempts to bind to a port that is already in use by another application, it will encounter an error, as the port is already occupied. To enable multiple applications to run on the same server, each application must use a different port number.

Q) Test your memory of the classes for reading and writing from a SocketChannel.

Ans:

To read text from a SocketChannel- Byte Buffer

To send text to a SocketChannel- ByteBuffer

Q) What two pieces of information does the client need in order to make a connection with a server?

Ans: IP Adress and port number

Q) Which TCP port numbers are reserved for "well-known services" like HTTP and FTP?

Ans:

HTTP-80

FTP-21

HTTPS-443

Q) TRUE or FALSE: The range of valid TCP port numbers can be represented by a short primitive

Ans: False

THERE ARE NO DUMB QUESTIONS(Pg: 621)

Q) Should I use a lambda expression for my Runnable or create a new class that implements Runnable?

Ans: The choice between using a lambda expression or creating a new class that implements `Runnable` depends on the complexity and reusability of the task. If the task is simple and consists of a small block of code, a lambda expression is often more concise and readable. It provides a streamlined way to define and pass the runnable code without the overhead of a separate class. However, if the task is complex, involves multiple methods, or needs to be reused across different parts of the application, creating a new class that implements `Runnable` might be more appropriate. This approach offers better organization and maintainability for more intricate logic.

Q) What's the advantage of using an ExecutorService? So far, it works the same as creating a Thread and starting it.

Ans: Using an `ExecutorService` offers several advantages over directly creating and starting threads. It provides a higher-level API for managing and controlling thread execution, allowing for more efficient and scalable handling of concurrent tasks. An `ExecutorService` can manage a pool of threads, which helps in reusing threads for multiple tasks, reducing the overhead of creating new threads. It also offers methods for scheduling tasks, submitting tasks for execution, and gracefully shutting down the service. This approach leads to better resource management, improved performance, and easier implementation of complex concurrency patterns compared to manually handling threads.

EXERCISE(Pg: 631)

Q) A bunch of Java and network terms, in full costume, are playing a party game, "Who am I?" They'll give you a clue—you try to guess who they are based on what they say. Assume they always tell the truth about themselves. If they happen to say something that could be true for more than one attendee, then write down all for whom that sentence applies. Fill in the blanks next to the sentence with the names of one or more attendees.

Ans:

I need to be shut down or I might live forever-ExecutorService

I let you talk to a remote machine-SocketChannel, Socket

I might be thrown by sleep() and await()-InterruptedException

If you want to reuse Threads, you should use me-Thread pool, ExecutorService

You need to know me if you want to connect to another machine-IP Address, Host name, port

I'm like a separate process running on the machine-Thread

I can give you the ExecutorService you need-Executors

You need one of me if you want clients to connect to me-ServerSocketChannel

I can help you make your multithreaded code more predictable-Thread.sleep(), CountDownLatch

I represent a job to run-Runnable

I store the IP address and port of the server-InetSocketAddress

CODE MAGNETS(Pg:634)

```
import java.io.*;

import java.net.InetSocketAddress;

import java.nio.channels.*;

import java.time.format.FormatStyle;

import java.util.concurrent.TimeUnit;

import static java.nio.charset.StandardCharsets.UTF_8;

import static java.time.LocalDateTime.now;

import static java.time.format.DateTimeFormatter.ofLocalizedTime;

public class PingingClient {

public static void main(String[] args) {

InetSocketAddress server = new InetSocketAddress("127.0.0.1", 5000);

try (SocketChannel channel = SocketChannel.open(server)) {

PrintWriter writer = new PrintWriter(Channels.newWriter(channel, UTF_8));

System.out.println("Networking established");

for (int i = 0; i < 10; i++) {

String message = "ping " + i;

writer.println(message);

writer.flush();
```

```
String currentTime = now().format(ofLocalizedTime(FormatStyle.MEDIUM));

System.out.println(currentTime + " Sent " + message);

TimeUnit.SECONDS.sleep(1);

}

} catch (IOException | InterruptedException e) {

e.printStackTrace();

}

}

}
```

CHAPTER 18 – DEALING WITH CONCURRENCY ISSUES

THERE ARE NO DUMB QUESTIONS

Pg: 652

Q) Sounds like it's a good idea to synchronize everything, just to be thread-safe.

Ans: It is not a good idea to synchronize everything just to achieve thread safety. While synchronization can prevent concurrent access issues, overuse can lead to significant performance drawbacks, such as reduced throughput and increased contention, leading to potential deadlocks and bottlenecks. Instead, synchronization should be used selectively and strategically to protect only the critical sections of code that access shared mutable state. Proper design patterns, such as using concurrent collections or atomic variables, can also help achieve thread safety more efficiently without the need to synchronize everything.

Pg: 660

Q) What happens if other parts of the program have a reference to the old Address?

Ans: If other parts of the program have a reference to the old `Address` object after it has been modified or replaced, they will still hold the reference to the old object and not see the updated or new `Address`. This can lead to inconsistencies and bugs if the program relies on the latest state of the `Address` object. To avoid such issues, it is important to ensure that all references to the `Address` object are updated appropriately or that the object is immutable, so any changes result in a new object rather than modifying the existing one. This way, consistency is maintained throughout the program.

THERE ARE NO DUMB QUESTIONS(Pg: 666)

Q) Doesn't using CopyOnWriteArrayList mean that some reading threads will be reading old data?

Ans: Using `CopyOnWriteArrayList` can indeed mean that some reading threads will read old data, but this is by design and is considered acceptable within its use cases. `CopyOnWriteArrayList` is optimized for scenarios where read operations significantly outnumber write operations. When a write operation occurs, a new copy of the array is created, and subsequent reads will access the new array. However, reads that started before the write operation may still access the old array until they complete. This provides a consistent view of the data at the cost of potentially reading slightly outdated information, which is usually acceptable in applications where the frequency of modifications is low and the consistency of read operations is more critical.

Q: **Isn't it bad to be using out of date data?**

Ans: Using out-of-date data can be problematic if real-time accuracy is critical. However, in many cases, a slight delay in data consistency is acceptable for better performance and scalability. `CopyOnWriteArrayList` is ideal for situations with infrequent writes and frequent reads, offering a consistent snapshot of data at the time of the read, which is sufficient for many applications. The balance between data freshness and performance should be considered based on the application's needs.

Q: **But I don't want my bank statement to be even slightly out of date! How can I make sure that critical shared data is always correct?**

Ans: For critical shared data that must always be correct and up-to-date, such as a bank statement, synchronization mechanisms should be used to ensure data consistency. This can be achieved by using synchronized blocks or methods to control access to shared data, preventing concurrent modifications. Alternatively, concurrent data structures like `ConcurrentHashMap` or using locks (e.g., `ReentrantLock`) can provide fine-grained control over data access and

modifications. These approaches ensure that data is always consistent and up-to-date, though they may come with some performance overhead due to the increased synchronization.

Q: So CopyOnWriteArrayList is a fast thread-safe data structure?

Ans: `CopyOnWriteArrayList` is a thread-safe data structure optimized for scenarios with many read operations and few write operations. It provides fast and consistent reads since readers do not need to lock or synchronize. However, write operations (add, set, remove) are relatively slow because they involve copying the entire array. Therefore, `CopyOnWriteArrayList` is ideal for use cases where reads significantly outnumber writes, offering excellent read performance with thread safety.

Q: Why isn’t there an easy answer to the best way to do this concurrency stuff?

Ans: There isn't an easy answer to the best way to handle concurrency because the optimal approach depends on the specific requirements and context of the application. Factors such as the frequency of read and write operations, the need for real-time data consistency, performance considerations, and the complexity of the data structures involved all influence the choice of concurrency mechanisms. Different scenarios may require different solutions, such as synchronized blocks, concurrent collections, lock mechanisms, or other concurrency control techniques, making it essential to carefully evaluate and choose the most appropriate strategy for each case.

Q: **You never told us about the case where adding final to a field declaration is not enough to make sure that value is never changed. What’s the deal?**

Ans: Adding the `final` keyword to a field declaration ensures that the field's value cannot be reassigned after it has been initialized. However, if the field is a reference to a mutable object, the object's internal state can still be changed.

For example, a `final` reference to a list can prevent the list reference from being reassigned, but the contents of the list can still be modified. To ensure complete immutability, the referenced

object itself must be immutable, meaning its state cannot be altered after it is created. This requires designing the object's class to prevent any changes to its internal state.

EXERCISE(pg 668)

Q) The Java file on this page represents a complete source file. Your job is to play JVM and determine what the output would be when the program runs. How might you fix it, making sure the output is correct every time?

Ans: There are two ways:

1) Synchronize the write method:

```
public synchronized void addLetter(char letter) {
letters.add(String.valueOf(letter));
}
```

2) Use a thread-safe collection:

```
private final List<String> letters = new CopyOnWriteArrayList<>();
```