Due to the print book page limit, we cannot inlcude all good CheckPoint questions in the physical book. The CheckPoint on this Website may contain extra questions not printed in the book. The questions in some sections may have been reordered as a result. Nevertheless, it is easy to find the CheckPoint questions in the book on this Website. Please send suggestions and errata to Dr. Liang at y.daniel.liang@gmail.com. Indicate the book, edition, and question number in your email. Thanks!

Chapter 11 Check Point Questions

Section 11.2

▼11.2.1

True or false? A subclass is a subset of a superclass.

False.
A subclass is an extension of a superclass and normally contains more details information than its superclass.

Hide Answer

▼11.2.2

What keyword do you use to define a subclass?

The extends keyword is used to define a subclass that extends a superclass.

Hide Answer

▼11.2.3

What is single inheritance? What is multiple inheritance? Does Java support multiple inheritance?

Single inheritance allows a subclass to extend only one superclass. Multiple inheritance allows a subclass to extend multiple classes. Java does not allow multiple inheritance.

Hide Answer

Section 11.3

▼11.3.1

What is the output of running the class C in (a)? What problem arises in compiling the program in (b)?

```
(a)
class A {
  public A() {
    System.out.println("A's no-arg constructor is invoked");
  }
}

class B extends A {
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}

(b)
```

```
class A {
  public A(int x) {
  }
}

class B extends A {
  public B() {
  }
}

public class C {
  public static void main(String[] args) {
    B b = new B();
  }
}
```

(a) The output is

```
            A's no-arg constructor is invoked
```

(b) The default constructor of B attempts to invoke the default of constructor of A, but class A's default constructor is not defined.

Hide Answer

▼11.3.2

How does a subclass invoke its superclass's constructor?

A subclass can explicitly invoke a suplerclass's constructor using the super keyword.

Hide Answer

▼11.3.3

True or false? When invoking a constructor from a subclass, its superclass's no-arg constructor is always invoked.

False.
If a subclass's constructor explicitly invoke a superclass's constructor, the superclass's no-arg constructor is not invoked.

Hide Answer

Section 11.4

▼11.4.1

True or false? You can override a private method defined in a superclass.

False.
You can only override accessible instance methods.

Hide Answer

▼11.4.2

True or false? You can override a static method defined in a superclass.

False.
You can only override accessible instance methods.

Hide Answer

▼11.4.3
How do you explicitly invoke a superclass's constructor from a subclass?

Use super() or super(args). This statement must be the first in the constructor in the subclass.

Hide Answer

▼11.4.4
How do you invoke an overridden superclass method from a subclass?

Use super.method() or super.method(args).

Hide Answer

Section 11.5

▼11.5.1
Identify the problems in the following code:

```java
public class Circle {
  private double radius;

  public Circle(double radius) {
    radius = radius;
  }

  public double getRadius() {
    return radius;
  }

  public double getArea() {
    return radius * radius * Math.PI;
  }
}

class B extends Circle {
  private double length;

  B(double radius, double length) {
    Circle(radius);
    length = length;
  }

  @Override
  public double getArea() {
    return getArea() * length;
  }
}
```

The following lines are erroneous:

```java
{
  radius = radius; // Must use this.radius = radius
}

class B extends Circle {
  Circle(radius);  // Must use super(radius)
```

```
    length = length; // Must use this.length = length
  }

  public double getArea() {
    return getArea() * length; // super.getArea()
  }
```

Hide Answer

▼**11.5.2**
Explain the difference between method overloading and method overriding.

Method overloading defines methods of the same name in a class. Method overriding modifies the methods that are defined in the superclasses.

Hide Answer

▼**11.5.3**
If a method in a subclass has the same signature as a method in its superclass with the same return type, is the method overridden or overloaded?

It is method overridden.

Hide Answer

▼**11.5.4**
If a method in a subclass has the same signature as a method in its superclass with a different return type, will this be a problem?

It will be a problem if the return type is not a compatible return type.

Hide Answer

▼**11.5.5**
If a method in a subclass has the same name as a method in its superclass with different parameter types, is the method overridden or overloaded?

It is method overloading.

Hide Answer

▼**11.5.6**
What is the benefit of using the @Override annotation?

It forces the compiler to check the signature of the overridden method to ensure that the method is defined correctly.

Hide Answer

Section 11.7

▼**11.7.1**
What are the three pillars of object-oriented programming? What is polymorphism?

Encapsulation, inheritance, and polymorphism. In simple terms, polymorphism means that a variable of a supertype can refer to a subtype object.

Hide Answer

Section 11.8

▼11.8.1

What is dynamic binding?

A method may be implemented in several classes along the inheritance chain. The JVM decides which method is invoked at runtime. This is known as dynamic binding.

Hide Answer

▼11.8.2

Describe the difference between method matching and method binding.

Matching a method signature and binding a method implementation are two separate issues. The declared type of the reference variable decides which method to match at compile time. The compiler finds a matching method according to parameter type, number of parameters, and order of the parameters at compile time. A method may be implemented in several subclasses. The JVM dynamically binds the implementation of the method at runtime, decided by the actual class of the object referenced by the variable.

Hide Answer

▼11.8.3

Can you assign new int[50], new Integer[50], new String[50], or new Object[50], into a variable of Object[] type?

new int[50] cannot be assigned to into a variable of Object[] type, but new Integer[50], new String[50], or new Object[50] are fine.

Hide Answer

▼11.8.4

What is wrong in the following code?

```
 1  public class Test {
 2    public static void main(String[] args) {
 3      Integer[] list1 = {12, 24, 55, 1};
 4      Double[] list2 = {12.4, 24.0, 55.2, 1.0};
 5      int[] list3 = {1, 2, 3};
 6      printArray(list1);
 7      printArray(list2);
 8      printArray(list3);
 9    }
10
11    public static void printArray(Object[] list) {
12      for (Object o: list)
13        System.out.print(o + " ");
14      System.out.println();
15    }
16  }
```

Line 8 causes a compile error, because int[] cannot be passed to Object[].

Hide Answer

▼11.8.5

Show the output of the following code:

```
(a)
public class Test {
  public static void main(String[] args) {
    new Person().printPerson();
    new Student().printPerson();
  }
}

class Student extends Person {
  @Override
  public String getInfo() {
    return "Student";
  }
}

class Person {
  public String getInfo() {
    return "Person";
  }

  public void printPerson() {
    System.out.println(getInfo());
  }
}

(b)
public class Test {
  public static void main(String[] args) {
    new Person().printPerson();
    new Student().printPerson();
  }
}

class Student extends Person {
  private String getInfo() {
    return "Student";
  }
}

class Person {
  private String getInfo() {
    return "Person";
  }

  public void printPerson() {
    System.out.println(getInfo());
  }
}
```

```
(a)
Person
Student

(b)
Person
Person
```

For (a), new Person().printPerson() invokes the printPerson() method definedin the Person class, which then invokes the getInfo() method in the Person class.new Student().printPerson() invokes the printPerson() method definedin the Person class, which then invokes the getInfo() method in the Student class, because the calling object is a Student.

For (b), new Student().printPerson() invokes the printPerson() method definedin the Person class, which then invokes the getInfo() method in the Person class. It does not invoke the getInfo() method in the Student class, because it is private and can only be invoked from a method in the Student class. Note that the getInfo() method is not overridden because it is private. You can only override a non-private method such as getInfo() in (a).

Hide Answer

▼11.8.6

Show the output of following program:

```java
public class Test {
  public static void main(String[] args) {
    A a = new A(3);
  }
}

class A extends B {
  public A(int t) {
    System.out.println("A's constructor is invoked");
  }
}

class B {
  public B() {
    System.out.println("B's constructor is invoked");
  }
}
```

Is the no-arg constructor of Object invoked when new A(3) is invoked?

```
B's constructor is invoked
A's constructor is invoked
```

The default constructor of Object is invoked, when new A(3) is invoked. The Object's constructor is invoked before any statements in B's constructor are executed.

Hide Answer

▼11.8.7

Show the output of following program:

```java
public class Test {
  public static void main(String[] args) {
    new A();
    new B();
  }
}

class A {
  int i = 7;

  public A() {
    setI(20);
    System.out.println("i from A is " + i);
  }

  public void setI(int i) {
    this.i = 2 * i;
  }
}

class B extends A {
  public B() {
    System.out.println("i from B is " + i);
  }

  public void setI(int i) {
    this.i = 3 * i;
  }
}
```

```
i from A is 40
i from A is 60
i from B is 60
```

Hide Answer

▼11.8.8
Show the output of following program:

```
public class Test {
  public static void main(String[] args) {
    Apple a = new Apple();
    System.out.println(a);
    System.out.println("---------------");

    GoldenDelicious g = new GoldenDelicious(7);
    System.out.println(g);
    System.out.println("---------------");

    Apple c = new GoldenDelicious(8);
    System.out.println(c);
  }
}

class Apple {
  double weight;

  public Apple() {
    this(1);
    System.out.println("Apple no-arg constructor");
  }

  public Apple(double weight) {
    this.weight = weight;
    System.out.println("Apple constructor with weight");
  }

  @Override
  public String toString() {
    return "Apple: " + weight;
  }
}

class GoldenDelicious extends Apple {
  public GoldenDelicious() {
    this(5);
    System.out.println("GoldenDelicious non-arg constructor");
  }

  public GoldenDelicious(double weight) {
    super(weight);
    this.weight = weight;
    System.out.println("GoldenDelicious constructor with weight");
  }

  @Override
  public String toString() {
    return "GoldenDelicious: " + weight;
  }
}
```

```
Apple constructor with weight
Apple no-arg constructor
Apple: 1.0
---------------
Apple constructor with weight
GoldenDelicious constructor with weight
GoldenDelicious: 7.0
---------------
Apple constructor with weight
GoldenDelicious constructor with weight
GoldenDelicious: 8.0
```

Hide Answer

Section 11.9

▼11.9.1

Indicate true or false for the following statements:
(a) You can always successfully cast an instance of a subclass to a superclass.
(b) You can always successfully cast an instance of a superclass to a subclass.

(a) True. You can always successfully cast an instance of a subclass to a superclass. for example, casting apple to fruit is fine.
(b) False. You cannot always successfully cast an instance of a superclass to a subclass. For example, casting fruit to apple is not always succeful unless a fruit is an apple.

Hide Answer

▼11.9.2

For the GeometricObject and Circle classes in Listings 11.1 and 11.2, answer the following questions:
a. Assume are circle and object1 created as follows:

```
Circle circle = new Circle(1);
GeometricObject object1 = new GeometricObject();
```

Are the following Boolean expressions true or false?

```
(circle instanceof GeometricObject)
(object instanceof GeometricObject)
(circle instanceof Circle)
(object instanceof Circle)
```

b. Can the following statements be compiled?

```
Circle circle = new Circle(5);
GeometricObject object = circle;
```

c. Can the following statements be compiled?

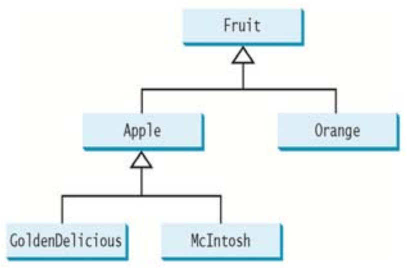
```
GeometricObject object = new GeometricObject();
Circle circle = (Circle)object;
```

```
(a)
(circle instanceof GeometricObject1) => true
(object instanceof GeometricObject1) => true
(circle instanceof Circle1) => true
(object instanceof Circle1) => false

(b)
Yes, because you can always cast from subclass to superclass.

(c)
Causing a runtime exception (ClassCastExcpetion)
```

Hide Answer

▼11.9.3

Suppose that Fruit, Apple, Orange, GoldenDelicious, and McIntosh are defined in the following inheritance hierarchy:



Assume that the following code is given:

```
Fruit fruit = new GoldenDelicious();
Orange orange = new Orange();
```

Answer the following questions:
a. Is fruit instanceof Fruit?
b. Is fruit instanceof Orange?
c. Is fruit instanceof Apple?
d. Is fruit instanceof GoldenDelicious?
e. Is fruit instanceof McIntosh?
f. Is orange instanceof Orange?
g. Is orange instanceof Fruit?
h. Is orange instanceof Apple?
i. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method?

j. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method?
k. Is the statement Orange p = new Apple() legal?
l. Is the statement McIntosh p = new Apple() legal?
m. Is the statement Apple p = new McIntosh() legal?

a. Is fruit instanceof Fruit true? true
b. Is fruit instanceof Orange true? false
c. Is fruit instanceof Apple true? true
d. Is fruit instanceof GoldDelicious true? true, because new GoldenDelicious() is assigned to fruit. The actual type for fruit at runtime is GoldenDelious.
e. Is fruit instanceof Macintosh true? false
f. Is orange instanceof Orange true? true
g. Is orange instanceof Fruit true? true
h. Is orange instanceof Apple true? false
i. Suppose the method makeApple is defined in the Apple class. Can fruit invoke this method? No. It will give a compile error. However, you can invoke it using ((Apple)fruit).makeAppleCider().
Can orange invoke this method? No
j. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Yes.
Can fruit invoke this method? No.
k. Is the statement Orange p = new Apple() legal? No
l. Is the statement Macintosh p = new Apple() legal? No
m. Is the statement Apple p = new Macintosh() legal? Yes

Hide Answer

▼11.9.4
What is wrong in the following code?

```
public class Test {
  public static void main(String[] args) {
    Object fruit = new Fruit();
    Object apple = (Apple)fruit;
  }
}

class Apple extends Fruit {
}

class Fruit {
}
```

```
Object apple = (Apple)fruit;
```

Causes a runtime ClassCastingException.

Hide Answer

Section 11.10

▼11.10.1
Does every object have a toString method and an equals method? Where do they come from? How are they used? Is it appropriate to override these methods?

Yes, because these two methods are defined in the Object class; therefore, they are available to all

Java classes. The subclasses usually override these methods to provide specific information for these methods.

The toString() method returns a string representation of the object; the equals() method compares the contents of two objects to determine whether they are the same.

Hide Answer

▼11.10.2
When overriding the equals method, a common mistake is mistyping its signature in the subclass. For example, the equals method is incorrectly written as equals(Circle circle), as shown in (a) in following the code; instead, it should be equals(Object circle), as shown in (b). Show the output of running class Test with the Circle class in (a) and in (b), respectively.

```
public class Test {
  public static void main(String[] args) {
    Object circle1 = new Circle();
    Object circle2 = new Circle();
    System.out.println(circle1.equals(circle2));
  }
}

(a)
class Circle {
  double radius;

  public boolean equals(Circle circle) {
    return this.radius == circle.radius;
  }
}

(b)
class Circle {
  double radius;

  public boolean equals(Object o) {
    return this.radius == ((Circle)o).radius;
  }
}
```

If Object is replaced by Circle in the Test class, what would be the output to run Test using the Circle class in (a) and (b), respectively?
Suppose that circle1.equals(circle2) is replaced by circle1.equals("Binding"), what would happen to run Test using the Circle class in (a) and (b), respectively? Reimplement the equals method in (b) to avoid a runtime error when comparing circle with a non-circle object.

The output is false if the Circle class in (a) is used. The Circle class has two overloaded methods: equals(Circle circle) defined in the Circle class and equals(Object o) defined in the Object class, inherited by the Circle class. At compile time, circle1.equals(circle2) is matched to equals(Object o), because the declared type for circle1 and circle2 is Object. (Note that either the declared type for circle1 and circle2 is Object would cause circle1.equals(circle2) to match circle1.equals(Object circle) by the compiler.

The output is true if the Circle class in (b) is used. The Circle class overrides the equals(Object o) method defined in the Object class. At compile time, circle1.equals(circle2) is matched to equals(Object o) and at runtime the equals(Object o) method implemented in the Circle class is invoked.

In (a), method equals(Object o) is used at both compile time and run time. circle1 and circle2 have different addresses, leading to "false" output. Method overriding follows dynamic binding (determined by the actual type), but method overloading is always determined by the declared type.

What would be the output if Object is replaced by Circle in the Test class using the Circle class in (a) and (b), respectively? The output would be true for (a), because circle1.equals(circle2) matches circle1.equals(Circle object) exactly in this case. The output would be true for (b) because equals(Object c) is overridden in the Circle class.

With circle1.equals(circle2) replaced by circle1.equals("Binding") and using the the Circle class in (a), the equals method in the Object class is used,circle1.equals(circle2) returns false.

With circle1.equals(circle2) replaced by circle1.equals("Binding") and using the the Circle class in (b), the equals method in the Circle class is used,since it casts string "Binding" to Circle, it throws a casting exception.
(b) should be reimplemented as follows to to avoid a runtime error when comparing circle with a non-circle object.

```
class Circle {
  double radius;

  public boolean equals(Object o) {
    if (o instanceof Circle)
      return this.radius == ((Circle)o).radius;
    else
      return false;
  }
}
```

Hide Answer

Section 11.11

▼11.11.1

How do you do the following?
a. Create an ArrayList for storing double values?
b. Append an object to a list?
c. Insert an object at the beginning of a list?
d. Find the number of objects in a list?
e. Remove a given object from a list?
f. Remove the last object from the list?
g. Check whether a given object is in a list?
h. Retrieve an object at a specified index from a list?

(a) ArrayList<Double> list = new ArrayList<Double>();
(b) list.add(object);
(c) list.add(0, object);
(d) list.size();
(e) list.remove(object);
(f) list.remove(list.size() - 1);
(g) list.contains(object);
(h) list.get(index);

Hide Answer

▼11.11.2

Identify the errors in the following code.

```
ArrayList<String> list = new ArrayList<>();
list.add("Denver");
list.add("Austin");
list.add(new java.util.Date());
String city = list.get(0);
list.set(3, "Dallas");
System.out.println(list.get(3));
```

Error 1: list.add(new java.util.Date()); is wrong, because it is an array list of strings. You cannot add Date objects to this list.
Error 2: list.set(3, "Dallas"); is wrong because there is no element at index 3 in the list.
Error 3: list.get(3) is wrong because there is no element at index 3 in the list.

Hide Answer

▼11.11.3

Suppose the ArrayList list contains {"Dallas", "Dallas", "Houston", "Dallas"}. What is the list after invoking list.remove("Dallas") one time? Does the following code correctly remove all elements with value "Dallas" from the list? If not, correct the code.

```
for (int i = 0; i < list.size(); i++)
  list.remove("Dallas");
```

After list.remove("Dallas"), the list becomes {"Dallas", "Houston", "Dallas"}. No. Here is the reason: Suppose the list contains two string elements "red" and "red". You want to remove "red" from the list. After the first "red" is removed, i becomes 1 and the list becomes {"red"}. i < list.size() is false. So the loop ends. The correct code should be

```
for (int i = 0; i < list.size(); i++) {
  if (list.remove(element))
```

```
    i--;
}
```

Hide Answer

▼11.11.4
Explain why the following code displays [1, 3] rather than [2, 3].

```
ArrayList<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.remove(1);
System.out.println(list);
```

How do you remove integer value 3 from the list?

The ArrayList class has two overloaded remove method remove(Object) and remove(int index). The latter is invoked for list.remove(1) to remove the element in the list at index 1.
To remove value 3 from the list, use list.remove(new Integer(3)).

Hide Answer

▼11.11.5
Explain why the following code is wrong.

```
ArrayList<Double> list = new ArrayList<>();
list.add(1);
```

list consists of Double objects. list.add(1) automatically converts 1 into an Integer object. It will work if you change it to list.add(1.0).

Hide Answer

Section 11.12

▼11.12.1
Correct errors in the following statements:

```
int[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(array));
```

To use asList(array), array must be an array of objects.

Hide Answer

▼11.12.2
Correct errors in the following statements:

```
int[] array = {3, 5, 95, 4, 15, 34, 3, 6, 5};
System.out.println(java.util.Collections.max(array));
```

To use Collections.max(array), array must be an ArrayList, not an array.

Hide Answer

Section 11.13

▼11.13.1

Write statements that create a MyStack and add number 11 to the stack.

MyStack stack = new MyStack();
stack.push(11); // 11 is autoboxed into new Integer(11)

Hide Answer

Section 11.14

▼11.14.1

What modifier should you use on a class so that a class in the same package can access it, but a class in a different package cannot access it?

default visibility modifier.

Hide Answer

▼11.14.2

What modifier should you use so that a class in a different package cannot access the class, but its subclasses in any package can access it?

protected

Hide Answer

▼11.14.3

In the following code, the classes A and B are in the same package. If the question marks in (a) are replaced by blanks, can class B be compiled? If the question marks are replaced by private, can class B be compiled? If the question marks are replaced by protected, can class B be compiled?

```
(a)
package p1;

public class A {
  ?  int i;

  ?  void m() {
    ...
  }
}

(b)
package p1;

public class B extends A {
  public void m1(String[] args) {
    System.out.println(i);
    m();
  }
}
```

If the question marks are replaced by blanks, can class B be compiled? Yes.
If the question marks are replaced by private, can class B be compiled? No.
If the question marks are replaced by protected, can class B be compiled? Yes.

Hide Answer

▼11.14.4

In the following code, the classes A and B are in different packages. If the question marks in (a) are replaced by blanks, can class B be compiled? If the question marks are replaced by private, can class B be compiled? If the question marks are replaced by protected, can class B be compiled?

```
(a)
package p1;

public class A {
  ?   int i;

  ?   void m() {
    ...
  }
}

(b)
package p2;

public class B extends A {
  public void m1(String[] args) {
    System.out.println(i);
    m();
  }
}
```

If the question marks are replaced by blanks, can class B be compiled? No.
If the question marks are replaced by private, can class B be compiled? No.
If the question marks are replaced by protected, can class B be compiled? Yes.

Hide Answer

▼11.14.5

In the following code, the classes A, B, and Main are in the same package. Can the Main class be compiled?

```
class A {
  protected void m() {
  }
}

class B extends A {
}

class Main {
  public void p() {
    B b = new B();
    b.m();
  }
}
```

Yes, because m() is protected. Even if A, B, and Main are different pakages, the Main class can be compiled, becasue the protected members can be accessed from a different package.

Hide Answer

Section 11.15

▼11.15.1

How do you prevent a class from being extended? How do you prevent a method from being overridden?

Use the final keyword.

Hide Answer

▼11.15.2

Indicate true or false for the following statements:
a. A protected datum or method can be accessed by any class in the same package.
b. A protected datum or method can be accessed by any class in different packages.
c. A protected datum or method can be accessed by its subclasses in any package.
d. A final class can have instances.
e. A final class can be extended.
f. A final method can be overridden.

a. True.
b. False. (But yes in a subclass that extends the class where the protected datum is defined.)
c. True.
d. Answer: True
e. Answer: False
f. Answer: False

Hide Answer