**Object Test Your Understanding**

1: If you don’t specify the parent class in a class declaration which of the following is true?

Top of Form

(A) It doesn't have a parent class.  
(B) It inherits from the Object class.  
(C) It inherits from the Default class.  
(D) It inherits from the Parent class.

2: If the class Vehicle has object fields of make and model and the class Car inherits from the class vehicle will a car object have a make and model?

Top of Form

(A) Yes  
(B) No

Bottom of Form

3: If I had a class ParkingGarage should it inherit from the class Vehicle?

Top of Form

(A) Yes  
(B) No  
Bottom of Form

4: In Java how many parents can a class have?

Top of Form

(A) 0  
(B) 1  
(C) 2  
(D) infinite  
Bottom of Form

1. If no parent class is specified using the extends keyword, the parent class will be Object.
2. Yes, a child class inherits all the parent class object field and methods.
3. No, a parking garage is not a kind of vehicle. Instead it has vehicles in it which implies that the ParkingGarage class would have a field that tracks the vehicles in it.
4. All classes in Java have one and only one parent class.

**2 .Check your understanding**

2-1: A bookstore is working on an on-line ordering system. For each type of published material (books and movies) they need to track the id, title, author(s), date published, and price. Which of the following would be the best design?

Top of Form

(A) Create one class PublishedMaterial with the requested fields plus type  
(B) Create classes Book and Movie and each class has the requested fields  
(C) Create the class PublishedMaterial and have Book and Movie inherit from it all the listed fields  
(D) Create one class BookStore with the requested fields plus type  
(E) Create classes for PublishedMaterial, Books, Movies, Title, Price, ID, Authors, DatePublished

2-2: A movie theater has multiple showings of a movie each day. Each movie showing has a start time and location (theater number). What should the relationship be between the Movie class and the MovieShowing class?

Top of Form

(A) The MovieShowing class should be a subclass of the Movie class.  
(B) The Movie class should be a subclass of the MovieShowing class.  
(C) A MovieShowing has a movie associated with it, so it should have a Movie field.  
Bottom of Form

2-3: What Java keyword is used to specify the parent class?

Top of Form

(A) superclass  
(B) parent  
(C) extends  
(D) class  
Check MeCompare me

Bottom of Form

**4: Question: Which of the following reasons for using an inheritance heirarchy are valid?**

1. Object methods from a superclass can be used in a subclass without rewriting or copying code.
2. Objects from subclasses can be passed as arguments to a method that takes an argument of the parent type.
3. Objects from subclasses can be stored in the same array of the parent type.
4. All of the above
5. None of the above

Top of Form

(A) V  
(B) IV  
(C) I and II  
(D) I and III  
(E) I onlyBottom of Form

1. C We will need to get objects based on their type so we should create classes for Book and Movie. They have common fields so we should put these in a common superclass PublishedMaterial.
2. C.  A movie showing is not a type of movie and a movie is not a type of movie showing. A movie showing has a movie associated with it.
3. C The extends keyword is used to specify the parent class.
4. B  All of these are valid reasons to use an inheritance heirarchy.

**3. Check your understanding**

3-1: Which of the following declarations in Student would correctly *override* the getFood method in Person?

**public** **class** **Person**

{

**private** String name = **null**;

**public** Person(String theName)

{

name = theName;

}

**public** String getFood()

{

**return** "Hamburger";

}

}

**public** **class** **Student** **extends** Person

{

**private** int id;

**private** **static** int nextId = 0;

**public** Student(String theName)

{

**super**(theName);

id = nextId;

nextId++;

}

**public** int getId() {**return** id;}

**public** void setId (int theId)

{

**this**.id = theId;

}

}

1. Top of Form

(A) public void getFood()  
(B) public String getFood(int quantity)  
(C) public String getFood()

1. Bottom of Form

10-3-2: Which of the following declarations in Person would correctly *overload* the getFood method in Person?

**public** **class** **Person**

{

**private** String name = **null**;

**public** Person(String theName)

{

name = theName;

}

**public** String getFood()

{

**return** "Hamburger";

}

}

**public** **class** **Student** **extends** Person

{

**private** int id;

**private** **static** int nextId = 0;

**public** Student(String theName)

{

**super**(theName);

id = nextId;

nextId++;

}

**public** int getId() {**return** id;}

**public** void setId (int theId)

{

**this**.id = theId;

}

}

1. Top of Form

(A) public void getFood()  
(B) public String getFood(int quantity)  
(C) public String getFood()

1. Bottom of Form

3-1: C The return type and parameter lists must match.

3-2 **B** For overloading you must change the parameter list (number, type, or order of parameters).

4-1: Given the following class declarations, and assuming that the following declaration appears in a client program: Base b = new Derived();, what is the result of the call b.methodOne();?

**public** **class** **Base**

{

**public** void methodOne()

{

System.out.print("A");

methodTwo();

}

**public** void methodTwo()

{

System.out.print("B");

}

}

**public** **class** **Derived** **extends** Base

{

**public** void methodOne()

{

**super**.methodOne();

System.out.print("C");

}

**public** void methodTwo()

{

**super**.methodTwo();

System.out.print("D");

}

}

**A incorrect** This would be true if the object was created of type Base using new Base. But the object is really a Derived object. So all methods are looked for starting with the Derived class.

**B correct** Even though b is declared as type Base it is created as an object of the Derived class, so all methods to it will be resolved starting with the Derived class. So the methodOne() in Derived will be called. This method first calls super.methodOne so this will invoke the method in the superclass (which is Base). So next the methodOne in Base will execute. This prints the letter "A" and invokes this.methodTwo(). Since b is really a Derived object, we check there first to see if it has a methodTwo. It does, so execution continues in Derived's methodTwo. This method invokes super.methodTwo. So this will invoke the method in the super class (Base) named methodTwo. This method prints the letter "B" and then returns. Next the execution returns from the call to the super.methodTwo and prints the letter "D". We return to the Base class methodOne and return from that to the Derived class methodOne and print the letter "C".

**C** Incorrect. After the call to methodOne in the super class printing "A", the code continues with the implicit this.methodTwo which resolves from the current object's class which is Derived. methodTwo in the Derived class is executed which then calls super.methodTwo which invokes printin "B" from methodTwo in the Base class. Then the "D" in the Derive methodTwo is printed. Finally the program returns to methodOne in the Derived class are prints "C".

**D** Incorrect. The call to methodTwo in super.methodOne is to this.methodTwo which is the method from the Derived class. Consequently the "D" is also printed.

**E** Incorrect. This is not an example of recursion. No method is called from within itself.

**Check your understanding**

5-1: Given the following class definitions which of the following would not compile if it was used in place of the missing code in the main method?

**class** **Item**

{

**private** int x;

**public** void setX(int theX) { x = theX; }

*// ... other methods not shown*

}

**public** **class** **EnhancedItem** **extends** Item

{

**private** int y;

**public** void setY(int theY) { y = theY; }

*// ... other methods not shown*

**public** **static** void main(String[] args)

{

EnhancedItem currItem = **new** EnhancedItem();

*// missing code*

}

}

(A) currItem.setX(3);  
(B) currItem.setY(2);  
(C) currItem.x = 3;  
(D) currItem.y = 2;

**A** Incorrect. The object currItem is an EnhancedItem object and it will inherit the public setX method from Item.

**B** Incorrect. The object currItem is an EnhancedItem object and that class has a public setY method.

**C Correct!** Even though an EnhancedItem object will have a x field the subclass does not have direct access to a private field. Use the public setX method instead.

D Incorrect. All code in the same class has direct access to all object fields.

6-1: Given the class definitions of Point2D and Point3D below, which of the constructors that follow (labeled I, II, and III) would be valid in the Point3D class?

**class** **Point2D** {

**public** int x;

**public** int y;

**public** Point2D() {}

**public** Point2D(int x,int y) {

**this**.x = x;

**this**.y = y;

}

*// other methods*

}

**public** **class** **Point3D** **extends** Point2D

{

**public** int z;

*// other code*

}

*// possible constructors for Point3D*

I. **public** Point3D() {}

II. **public** Point3D(int x, int y, int z)

{

**super**(x,y);

**this**.z = z;

}

III. **public** Point3D(int x, int y)

{

**this**.x = x;

**this**.y = y;

**this**.z = 0;

}

2: Given the class definitions of Point and NamedPoint below, which of the constructors that follow (labeled I, II, and III) would be valid in the NamedPoint class?

**class** **MPoint**

{

**private** int myX; *// coordinates*

**private** int myY;

**public** MPoint( )

{

myX = 0;

myY = 0;

}

**public** MPoint(int a, int b)

{

myX = a;

myY = b;

}

*// ... other methods not shown*

}

**public** **class** **NamedPoint** **extends** MPoint

{

**private** String myName;

*// constructors go here*

*// ... other methods not shown*

}

*// Proposed constructors for this class:*

I. **public** NamedPoint()

{

myName = "";

}

II. **public** NamedPoint(int d1, int d2, String name)

{

myX = d1;

myY = d2;

myName = name;

}

III. **public** NamedPoint(int d1, int d2, String name)

{

**super**(d1, d2);

myName = name;

}

(A) I only  
(B) I and III  
(C) II only  
(D) III only

**6-1**

A Incorrect. I is true because Point2D does have a no-arg constructor. II is true because Point2D does have a constructor that takes x and y. III is true because Point2D does have a no-arg constructor which will be called before the first line of code is executed in this constructor. The fields x and y are public in Point2D and thus can be directly accessed by all classes.

B Incorrect. Point2D does have a constructor that takes an x and y value so this is okay. Also the call to super is the first line of code in the child constructor as required. However, both I and III are okay as well.

C Incorrect. The x and y values in Point2D are public and so can be directly accessed by all classes including subclasses. Also there is a no-arg constructor in Point2D so the super no-arg constructor will be called before the first line of code in this constructor.

D Correct! I is true because Point2D does have a no-arg constructor. II is true because Point2D does have a constructor that takes x and y. III is true because Point2D does have a no-arg constructor which will be called before the first line of code is executed in this constructor. The fields x and y are public in Point2D and thus can be directly accessed by all classes.

6-2. A Incorrect. I is okay but III is also okay.

B Correct! NamedPoint will inherit from Point all fields but the fields are private and they can not be directly accessed in NamedPoint. You can use super as the first line in a constructor to initialize inherited fields. You can also set your own fields in a constructor. If you don't use super as the first line in a constructor one will be put there by the compiler that will call the parent's no argument constructor.

C Incorrect. II is invalid. Children inherit all of the fields from a parent but do not have direct access to private fields. You can use super in a constructor to initialize inherited fields by calling the parent's constructor with the same parameter list.

D Incorrect. I is also okay

8-1: Which of the following is true about interfaces?

I. Interfaces can only contain **abstract** methods or **class** **constants**.

II. Interfaces can be extended.

III. Interfaces can be instantiated (you can create an object of the **interface** **type**).

Top of Form

(A) I only  
(B) II only  
(C) I and II  
(D) I, II, and III

A Incorrect. Interfaces can also be extended (inherited from).

B Incorrect. II is true, but I is also true.

C Correct! Both I and II are true.

D incorrect. You can not create an object of an interface type. You can only create objects from concrete (not abstract) classes.

**Check your understanding**

10-9-2: What is the output from running the main method in the GradStudent class?

**public** **class** **Student** {

**public** String getFood() {

**return** "Pizza";

}

**public** String getInfo() {

**return** **this**.getFood();

}

**public** **static** void main(String[] args)

{

Student s1 = **new** GradStudent();

s1.getInfo();

}

}

**class** **GradStudent** **extends** Student {

**public** String getFood() {

**return** "Taco";

}

}

Top of Form

(A) Pizza  
(B) Taco  
(C) You will get a compile time error  
(D) You will get a run-time error  
Check MeCompare me

Bottom of Form

You can step through this code using the Java Visualizer by clicking on the following link [Student Example](http://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+Student+%7B%0A+++%0A+++public+String+getFood()+%7B%0A++++++return+%22Pizza%22%3B%0A+++%7D%0A+++%0A+++public+String+getInfo()++%7B+%0A++++++return+this.getFood()%3B+%0A+++%7D%0A+++%0A+++public+static+void+main(String%5B%5D+args)%0A+++%7B%0A++++++Student+s1+%3D+new+GradStudent()%3B%0A++++++System.out.println(s1.getInfo())%3B%0A+++%7D%0A%7D%0A%0Aclass+GradStudent+extends+Student+%7B%0A+++%0A+++public+String+getFood()+%7B%0A++++++return+%22Taco%22%3B%0A+++%7D%0A++++++++%0A+++%0A%7D%0A&mode=display&curInstr=10).

10-9-3: What is the output from running the main method in the RaceCar class?

**public** **class** **Car**

{

**private** int fuel;

**public** Car() { fuel = 0; }

**public** Car(int g) { fuel = g; }

**public** void addFuel() { fuel++; }

**public** void display() { System.out.print(fuel + " "); }

**public** **static** void main(String[] args)

{

Car car = **new** Car(5);

Car fastCar = **new** RaceCar(5);

car.display();

car.addFuel();

car.display();

fastCar.display();

fastCar.addFuel();

fastCar.display();

}

}

**class** **RaceCar** **extends** Car

{

**public** RaceCar(int g) { **super**(2\*g); }

}

Top of Form

(A) 5 6 10 11  
(B) 5 6 5 6  
(C) 10 11 10 11  
(D) The code won't compile.  
Check MeCompare me

Bottom of Form

You can step through the code using the Java Visualizer by clicking on the following link: [Car Example](http://cscircles.cemc.uwaterloo.ca/java_visualize/#code=public+class+Car%0A%7B%0A+++private+int+fuel%3B%0A%0A+++public+Car()+%7B+fuel+%3D+0%3B+%7D+%0A+++public+Car(int+g)+%7B+fuel+%3D+g%3B+%7D%0A%0A+++public+void+addFuel()+%7B+fuel%2B%2B%3B+%7D%0A+++public+void+display()+%7B+System.out.print(fuel+%2B+%22+%22)%3B+%7D%0A+++%0A+++public+static+void+main(String%5B%5D+args)%0A+++%7B%0A++++++Car+car+%3D+new+Car(5)%3B%0A++++++Car+fastCar+%3D+new+RaceCar(5)%3B%0A++++++car.display()%3B%0A++++++car.addFuel()%3B%0A++++++car.display()%3B%0A++++++fastCar.display()%3B%0A++++++fastCar.addFuel()%3B%0A++++++fastCar.display()%3B%0A++++%7D%0A%7D%0A%0Aclass+RaceCar+extends+Car%0A%7B%0A+++public+RaceCar(int+g)+%7B+super(2*g)%3B+%7D%0A%7D+%0A&mode=display&curInstr=0).

10-9-4: Given the following class definitions and a declaration of Book b = new Dictionary which of the following will cause a compile-time error?

**public** **class** **Book**

{

**public** String getISBN()

{

*// implementation not shown*

}

*// constructors, fields, and other methods not shown*

}

**public** **class** **Dictionary** **extends** Book

{

**public** String getDefinition(String word)

{

*// implementation not shown*

}

}

Top of Form

(A) b.getISBN();  
(B) b.getDefintion();  
(C) ((Dictionary) b).getDefinition();  
Check MeCompare me

Bottom of Form

10-9-5: Assume that the following declaration appears in a client program **Base b = new Derived();**. What is the result of the call **b.methodOne()**?

**public** **class** **Base**

{

**public** void methodOne()

{

System.out.print("A");

methodTwo();

}

**public** void methodTwo()

{

System.out.print("B");

}

**public** **static** void main(String[] args)

{

Base b = **new** Derived();

b.methodOne();

}

}

**class** **Derived** **extends** Base

{

**public** void methodOne()

{

**super**.methodOne();

System.out.print("C");

}

**public** void methodTwo()

{

**super**.methodTwo();

System.out.print("D");

}

}

Top of Form

(A) ABDC  
(B) AB  
(C) ABCD  
(D) ABC  
9-1answer

A Incorrect. The Rectangle subclass of Shape overrides the what method so this can't be right.

B Incorrect. The Square subclass doesn't not override the what method so it will use the one in Rectangle.

C Incorrect. This code will compile. The declared type can hold objects of that type or any subclass of the type.

D Correct! The Shape object will print Shape. The Rectangle object will print Rectangle. The Square object will also print Rectangle since it doesn't overrride the what method. The Circle object will print Circle.

E Incorrect. The Circle class does override the what method so this can't be right.

9-2

A Incorrect. This would be true if s1 was actually a Student, but it is a GradStudent. Remember that the run-time will look for the method first in the class that created the object.

B Correct! Even though the getInfo method is in Student when getFood is called the run-time will look for that method first in the class that created this object which in this case is the GradStudent class.

C Incorrect. This code will compile. The student class does have a getInfo method.

D Incorrect. There is no problem at run-time.

9-3

A Correct! The code compiles correctly, and because RaceCar extends the Car class, all the public object methods of Car can be used by RaceCar objects.

B Incorrect. RaceCar, while it inherits object methods from Car via inheritance, has a separate and different constructor that sets the initial fuel amount to 2 \* g, thus in this case, fuel for fastCar is set to 10 initially.

C Incorrect. The variable car is a Car object, so the constructor used is not the same as the fastCar object which is a RaceCar. The car constructor does not change the passed in parameter, so it is set to 5 initially.

D Incorrect. RaceCar inherits from the Car class so all the public object methods in Car can be accessed by any object of the RaceCar class.

9-4

A Incorrect. The b object is actually a Dictionary object which inherits the getISBN method from Book

B Correct! At compile time the declared type is Book and the Book class does not have or inherit a getDefintion method.

C Incorrect. Casting to Dictionary means that the compiler will check the Dictionary class for the getDefinition method.

9-5

A Correct! Even though b is declared as type Base it is created as an object of the Derived class, so all methods to it will be resolved starting with the Derived class.

B Incorrect. This would be true if the object was created of type Base using new Base. But the object is really a Derived object. So all methods are looked for starting with the Derived class.

C Incorrect. After the call to methodOne in the super class printing "A", the code continues with the implicit this.methodTwo which resolves from the current object's class which is Derived. methodTwo in the Derived class is executed which then calls super.methodTwo which invokes printin "B" from methodTwo in the Base class. Then the "D" in the Derive methodTwo is printed. Finally the program returns to methodOne in the Derived class are prints "C".

D Incorrect. The call to methodTwo in super.methodOne is to this.methodTwo which is the method from the Derived class. Consequently the "D" is also printed.