## سوال ۱

خروجی این برنامه را بدست بیاورید و به ازای هر خط توضیح دهید که چرا به این خروجی رسید؟

**class** Classes {

**static class** A {

**static int** *intValue* = 0;

**int integerValue** = 20;

A() {

**integerValue** = 5;

printValue();

print();

}

**void** printCaller() {

print();

}

**void** printValue() {

System.***out***.println(**"B:"** + **integerValue**);

}

**void** print() {

System.***out***.println(**"A:"** + *intValue*);

}

}

**static class** B **extends** A {

B(**int** v) {

*intValue* = v;

**integerValue** = 15;

printValue();

print();

}

**void** print() {

System.***out***.println(**"B:"** + *intValue*);

}

**void** printSuper() {

**super**.print();

}

**void** printCaller() {

printValue();

**super**.printValue();

}

**void** printValue() {

System.***out***.println(**"B:"** + **integerValue**);

**super**.printValue();

}

}

**static public class** C **extends** A {

**void** printCaller() {

System.***out***.println(**"B:"** + **integerValue**);

}

**void** print() {

System.***out***.println(**"A:"** + *intValue*);

**super**.printCaller();

}

}

}

**class** Problem1 {

**public static void** incrementValue(Classes.A object) {

object.*intValue*++;

object.**integerValue**++;

}

**public static void** incrementValue(**int** firstValue, **int** secondValue) {

firstValue++;

secondValue++;

}

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

Classes.A a = **new** Classes.A();

Classes.B b = **new** Classes.B(10);

Classes.A c = b;

b.print();

c.print();

((Classes.A) b).print();

b.printSuper();

a.printCaller();

b.printCaller();

c.printCaller();

*incrementValue*(a);

a.printCaller();

*incrementValue*(b);

b.printCaller();

*incrementValue*(c);

c.printCaller();

*incrementValue*(b.*intValue*, b.**integerValue**);

b.printCaller();

c.printCaller();

}

}

## سوال ۲

توضیح دهید که هدف از ارث بری در شی گرایی چیست. چه زمان از composition و چه زمان از inheritance استفاده می‌کنیم؟ چگونه می‌توانیم از سازنده پدر را فراخوانی کنیم؟ چگونه می‌توانیم سازنده دیگری از خود کلاس را فراخوانی کنیم؟

با ارثبری میتوان کلاس های مرتب تری ساخت ینی کلاس هارا جوری تعریف کرد که به ساختار دنیای واقعی شبیه تر باشه که در آن سوپر کلاسها مشخصاتی دارند که بین تمام اشیا خود و زیرکلاسهایشان مشترک باشند. این باعث میشود که از کپی کردن کد پرهیز شود(ترکیب هم این فایده را دارد) و با تعریف کلاس های انتزاعی قوانینی برای تعریف زیرکلاسها مشخص کنیم.زمانی از ارثبری استفاده میکنیم که رابطه is a برقرار باشد  
زمانی از ترکیب استفاده میکنیم که رابطه has a برقرار باشد

مثلا هر شغل نیاز به کارمند دارد پس در تعریف کلاس شغل از اشیا کارمند استفاده میکنیم

یا هر یک از سگ، گربه، ماهی و … یک نوع حیوان هستند پس از سوپرکلاس حیوان ارث بری میکنند

## سوال ۳

توضیح دهید که چرا از رابط‌ها (interface) استفاده می‌کنیم. چه محدودیت‌هایی نسبت به یک کلاس دارند و چرا امکان پیاده‌سازی متد در آنها داده شده است؟

از رابط ها وقتی استفاده میشود که ما بخواهیم کلیات رفتار اشیا را مشخص کنیم یا طراحی یک سری از کلاسها را توصیف کنیم. یا گاهی اوقات صرفا برای تعریف یک پروتکل استفاده میشود و در آن متد انتزاعی ای نداریم.

در رابط ها ویژگی و متد تعریف نمیکنیم( در حالی که در کلاس انتزاعی این کار را ممکن بود انجام دهیم) البته اگر متغیری تعریف کنیم ان متغیر ثابت استاتیک و پابلیک خواهد بود و در جاوا هشت هم میتوان به شکلی متد هایی تعریف کرد. ویژگی خوبی که رابط ها دارند این است که یک کلاس میتواند از چند رابط استفاده کند که اگر رابط ها درست تعریف شده باشند هم مشکلی پیش نمی آید ولی کلاس ها فقط از یک کلاس (چه معمولی و چه انتزاعی ) میتوانند ارث بری کنند تا از تداخل سوپرکلاسها در رفتار و ویژگی های کلاس مورد نظر جلوگیری شود

## سوال ۴

کلاس انتزاعی (abstract) چیست و چه زمانی در مدل‌سازی از یک کلاس انتزاعی استفاده می‌کنیم؟ این نوع کلاس چه تفاوتی با رابط(interface) دارد؟

کلاس های انتزاعی کلاس هایی هستند که سازنده ندارند یعنی ما برای آنها شی تعریف نمیکنیم بلکه باید زیرکلاس هایی داشته باشند تا برای آنها شی تعریف شود. در این کلاسها میتوان متد هایی تعریف کرد که انتزاعی باشند و برنامه نویس باید آنها را در کلاسهایی که ازین کلاس ارث می‌برند بصورت کامل تعریف کند. تفاوتی که با رابط ها دارند این است که ما میتوانیم در یک کلاس انتزاعی متد هایی تعریف کنیم که انتزاعی نباشند و صرفا بخاطر انتزاعی بودن این کلاس برای آن شی تعریف نکنیم اما رابط ها بدون جزییات تعریف میشوند و در آن متد ها تنها بصورت بدون body نوشته میشوند. درمورد تفاوت در ارثبری هم در سوال بالا توضیح داده شده.

## سوال ۵

override کردن تابع و متغیر چه تاثیری در عملکرد متد در یک کلاس فرزند می‌گذارد؟ چطور می‌توانیم پس از override شدن یک متد در کلاس فرزند در هر کدام از مکان‌های زیر به نسخه هم نام آن متد در کلاس پدر دسترسی پیدا کنیم؟

* متدی داخل کلاس پدر
* متدی داخل کلاس فرزند
* خارج از دو کلاس

جواب:  
اینکار باعث میشود که تابع مورد نظر دیگر به صورتی که در کلاس پدر تعریف شده کار نکند و ما تعریف جدیدی از آن را برای زیرکلاس مورد نظر مشخص میکنیم.

برای صدا زدن آن در کلاس پدر متد را بدون پیشوند صدا میزنیم

برای صدا زدن نسخه پدر در کلاس فرزند از واژه super استفاده میکنیم

در خارج از این دو کلاس هم صدا زدن متد مربوطه به ويژگی هایی مثل استاتیک بودن متد ربط دارد. مثلا اگر متد پدر را برای یک شی از فرزند بخواهیم صدا کنیم باید بین نام شی و نام متد کلمه سوپر را هم بنویسیم. یا مثلا اگر بخواهیم یک متد استاتیک را صدا بزنیم صرفا بعد از اسم کلاس پدر اسم متد را مینویسیم

## سوال ۶

توضیح دهید که منظور از چندریختی در شی گرایی چیست و چه مزیتی ایجاد می‌کند.

## سوال ۷

چرا از توابع و متدها در زبان برنامه نویسی استفاده می‌کنیم؟ در طراحی برنامه و شکستن آن به توابع و متدهای مختلف چه نکته‌هایی را باید رعایت کرد که خوانایی آن بیشتر شود و پیچیدگی اضافی نداشته باشیم؟

## سوال ۸

کلاس درونی (inner class) چه انواعی دارد و هر کدام چه کاربردی در مدل‌سازی و توصیف موجودات دارد؟ چگونه می‌توانیم یک شی از هر نوع ایجاد کنیم؟ در صورت override شدن یک متد یا متغیر توسط یک کلاس درونی چگونه می‌توان به نسخه override شده از کلاس بیرونی دسترسی پیدا کرد؟

## سوال ۹

کلمه کلیدی final روی هر کدام از موارد زیر چه تاثیری دارد؟

* تابع و متد
* تعریف کلاس←
* یک متغیر از نوع شی← محل حافظه ارجاع داده شده تغییر نمیکند
* یک متغیر از نوع‌ پایه ← مقدار متغیر تغییر نمیکند

## سوال ۱۰

کلمه کلیدی static روی هر کدام از موارد زیر چه تاثیری دارد؟

* تابع و متد ← باعث میشود که تابع مربوط به کل کلاس و متغیر های استاتیک شود و با فراخوانی کلاس صدا زده میشود نه با فراخوانی شی
* تعریف کلاس ← کلاس استاتیک برای زیرکلاس ها تعریف میشود و باعث میشود که به ویژگی های غیر استاتیک کلاس دسترسی نداشته باشد . به عبارتی دیگر به شی ای از کلاس بیرونی اشاره نکند و به خود کلاس ارجاع داشته باشد
* یک متغیر از نوع شی← یک متغیر برای هر شی تعریف میشود
* یک متغیر از نوع‌ پایه← یک متغیر برای کل کلاس تعریف میشود