سوال اول)

الف) یکی از مزایای مهم ارث‌بری و استفاده از آن کم کردن code duplication است. زیرا با استفاده از ارث‌بری می‌توان کدهای تکراری موجود در کلاس‌های مربوط را در یک کلاس parent گذاشت و از آن‌ها استفاده کرد. همچنین استفاده از ارث‌بری باعث می‌شود که کد ما انعطاف بیشتری داشته باشد که این امر به خصوص در مواقعی که می‌خواهیم تغییراتی در کدمان به وجود بیاوریم به ما کمک می‌کند که با کمترین اعمال تغییرات این کار را انجام دهیم. همچنین در بسیاری از مواقع ارث‌بری باعث خوانایی بیشتر کد ما می‌شود و همچنین کار دیباگینگ را نیز برای ما راحت‌تر می‌کند.

ب) می‌توانیم کلاسی را تصور کنیم که از دو کلاس دیگر به صورت همزمان ارث‌بری می‌کند. حال اگر دو کلاس والد یک متود با یک نام داشته باشند و کلاس child بخواهد از این کلاس‌ها ارث‌بری کند، جاوا نمی‌تواند تشخیص بدهد که باید از کدام متود و با کدام پیاده‌سازی ارث‌بری کند. به همین دلیل استفاده از ارث‌بری چندگانه از چند کلاس در جاوا ممکن نیست.

ج) در زبان جاوا کلمه‌ی کلیدی super در واقع رفرنسی به اولین سوپرکلاسی که کلاسی که در آن از این keyword استفاده شده است. با استفاده از این keyword میتوان به constructor و همچنین متودهای سوپرکلاس یک ساب‌کلاس دسترسی پیدا کرد.

د) تفاوت abstraction با inheritance در این است که هدف و استفاده‌ی ما از abstraction در زمانی است که می‌خواهیم فقط functionality برای کاربر ما دردسترس باشد و نمی‌خواهیم که از نحوه‌ی پیاده‌سازی و یا هر نوع اطلاعات داخلی دیگری باخبر باشد و برایش قابل مشاهده باشد. این کار را در جاوا می‌توان با استفاده از امکاناتی مانند abstract class و یا interface انجام داد. اما از طرف دیگر inheritance در واقع به ما امکان استفاده از اطلاعات و متودهای یک کلاس که از پیش طراحی شده را می‌دهد تا در واقع از code duplication جلوگیری شود و فقط با extend کردن یک سوپرکلاس بتوان از اطلاعات و یا متودهای آن استفاده کرد.

ه) overloading در جاوا به این معنی است که ما در یک کلاس متودهایی را با نام یکسان ولی با پارامترهای ورودی متفاوت(در تعداد پارامتر، در نوع پارامترها یا در هر دو) قرار بدهیم. در عوض، اگر در سوپرکلاس یک ساب‌کلاس متودی را داشته باشیم و در ساب‌کلاسِ آن کلاس متودی را دقیقا با همان نام و همان signature پیاده کنیم، در واقع آن متود را override می‌کنیم. این کار به خصوص در مواقعی که از چندریختی استفاده می‌کنیم می‌تواند به ما کمک کند.

و) این کار در زبان جاوا امکان‌پذیر است و درواقع به سبب اینکه نحوه‌ی پیاده‌سازی آن دو متود نیز کاملا مطابق یکدیگر است برای کامپایلر نیز مشکلی برای تشخیص به وجود نمی‌اید زیرا در واقع تفاوتی بین آن دو نیز وجود ندارد. فقط باید دقت کرد که این دو متود فقط باید یکبار در کلاس هدف override شوند نه دو بار. همچنین باید دقت کرد که signature آن دو متود نیز کاملا شبیه به هم باشد مگر نه به ارور برخورد خواهیم کرد.

سوال دو)

بخش اول)

این مشکل در واقع ناشی از این موضوع است که ما در زبان جاوا نمی‌توانیم از ارث‌بری چندگانه استفاده کنیم به این دلیل که در صورت داشتن متودهایی مشابه در سوپرکلاس‌ها جاوا نمی‌تواند تشخیص بدهد که باید از کدام متود ارث‌بری کند. البته باید این را در نظر داشت که برای رئوس کناری در diamond problem مشکلی وجود ندارد و این راس پایین لوزی است که به دلیل ارث‌بری چندگانه باعث ایجاد ارور می‌شود. برای مثال می‌توان در نظر گرفت که کلاس A در راس پایین لوزی از دو کلاس B و C ارث می‌برد و هر دوی این کلاس ها از کلاسی به نام D ارث می‌برند. در نتیجه ما در دیاگرام کشیدن به شکلی مانند لوزی می‌رسیم که به همین دلیل به آن Diamond Problem می‌گوییم.

بخش دوم)

الف)

. درست. زیرا رفرنسی از کلاس Parent را در نظر گرفتیم و شی جدیدی از ساب‌کلاس آن به آن assign کرده‌ایم.

. غلط. زیرا داریم به رفرنسی از ساب‌کلاسِ کلاس TwoDimensionalShape یک شی از کلاس TwoDimensionalShape می‌دهیم در حالیکه عکس این کار ممکن است.

. غلط. زیرا کلاس Triangle از کلاس ThreeDimensionalShape ارث‌بری نمی‌کند و در واقع در سمت دیگری از chart مربوط به کلاس‌ها قرار دارد.

. درست. زیرا به یک رفرنس از سوپرکلاس شی جدیدی از ساب‌کلاس آن assign کرده‌ایم.

. غلط. زیرا داریم به رفرنسی از ساب‌کلاس کلاس shape شی جدیدی از سوپرکلاس آن assign می‌کنیم در حالیکه عکس این کار ممکن است.

(ب

. غلط. در ابتدا c1 رفرنسی با استاتیک‌تایپ Circle و به یک آبجکت از کلاس Circle بوده است و سپس به آن رفرنس t1 که استاتیک‌تایپ آن TwoDimensionalShape و و به یک آبجکت از کلاس Triangle بوده است اسایت کرده‌ایم. می‌دانیم که هر دو کلاس Triangle و Circle به صورت جداگانه از یک سوپرکلاس ارث‌بری می‌کنند و این دو کلاس در واقع با هم هیچ رابطه‌ی ارث‌بری ندارند پس یک رفرنس از یکی از این دو کلاس نمی‌تواند به شی از کلاس دیگر اشاره کند.

. درست. رفرنس سمت چپ رفرنسی به یک آبجکت از سوپر کلاسِ کلاسِ آبجکتِ سمت راست است و طبق قاعده، رفرنس سوپرکلاس می‌تواند به آبجکتی از ساب‌کلاسش اشاره کند.

. درست. S2 در واقع رفرنسی با استاتیک تایپ از کلاس Shape است که در عبارت سوم به آن رفرنسی به یک آبجکت از کلاس Tetrahedron که در واقع ساب‌کلاسِ کلاس Shape است assign کرده‌ایم و می‌دانیم که رفرنس سوپرکلاس می‌تواند شی از ساب‌کلاسش را نگه‌داری کند.

.غلط. می‌دانیم که t2 رفرنسی از نوع ThreeDimensionalShape است که در ابتدا نیز اشاره به شی از همین کلاس دارد. همچنین می‌دانیم که s3 در واقع رفرنسی با استاتیک‌تایپ Shape است که به شی از ساب‌کلاس آن یعنی Cube اشاره می‌کند. هنگامی که s3 را به t2 اساین می‌کنیم، در واقع داریم رفرنسی را از سوپرکلاسِ کلاس به رفرنسی از ساب‌کلاس آن یعنی ThreeDimensionalShap اساین می‌کنیم که می‌دانیم اشتباه است.

.غلط. هر دو رفرنس در واقع رفرنس‌هایی به دو آبجکت هستند که هر کدام به صورت جداگانه از یک سوپرکلاس ارث‌بری می‌کنند و بین این دو کلاس هیچ رابطه‌ی ارث‌بری مستقیمی وجود ندارد. پس امکان اساین کردن رفرنس‌هایی از این دو کلاس به یکدیگر وجود ندارد.

. غلط. درس سمت چپ رفرنسی به شی از کلاس ThreeDimensionalShape وجود دارد که داریم به آن یک رفرنس از نوع TwoDimensionalShape که در واقع اشاره به یک آبجکت از ساب‌کلاس آن یعنی Triangle دارد اساین می‌کنیم. می‌دانیم که دو رفرنس در واقع از کلاس‌هایی هستند که در دو سمت مختلف از دیاگرام ارث‌بری ما قرار دارند و رابطه‌ی ارث‌بری با یکدیگر ندارند، پس امکان برابر قرار دادن رفرنس‌ها وجود ندارد.

سوال سوم)

الف)

A) Protected
B) Constructor
C) Super
D) Private
E) Polymorphism
F) Abstract
G) Abstract
H) Interfaces
I) Private
J) Concrete

ب)

1- غلط
2- درست
3- درست
4- درست
5- غلط
6- درست
7- درست