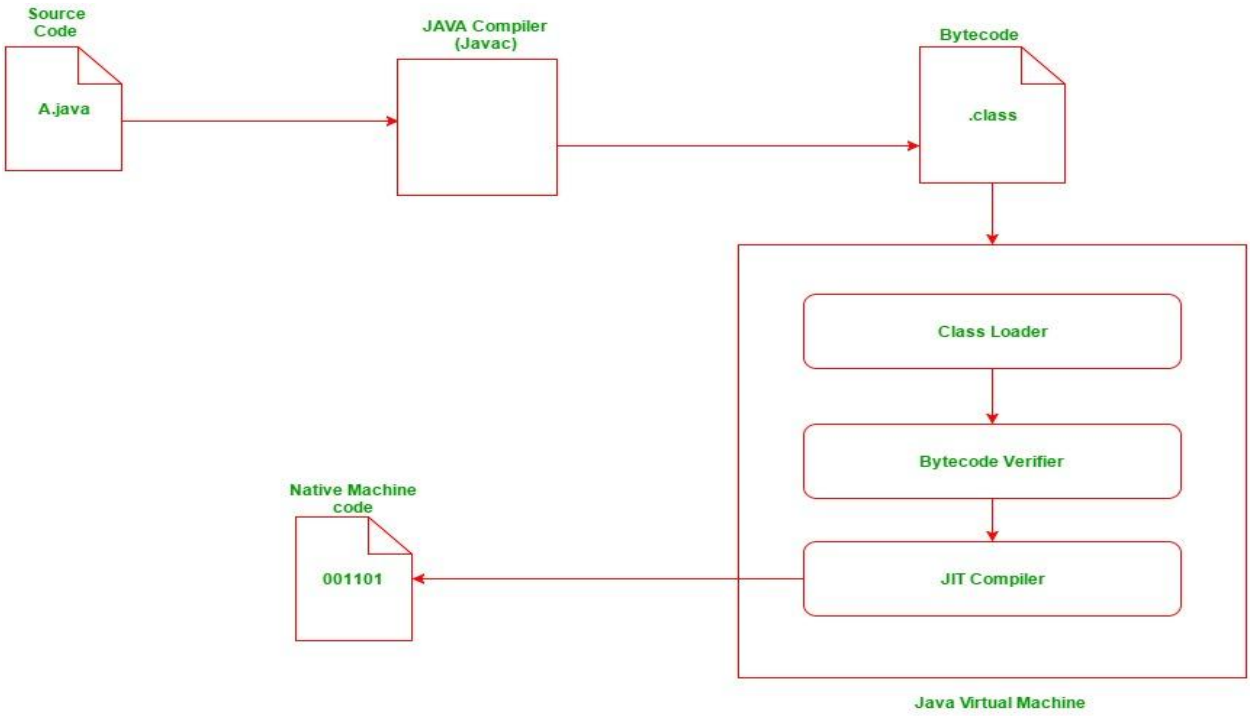
جواب قسمت الف :

فلسفه ی کلی در درست کردن زبان java به این صورت بود که اصلا به سیستم عاملی که کد در آن اجرا می شود بستگی نداشته باشد و کدی که با زبان java نوشته می شود دیگر هیچ نیازی به تغییر برای اجرا در سیستم عامل های دیگر نداشته باشد (اصطلاحا platform independent باشد) برای اینکار نیاز به ابزار هایی است که هر کدام از آنها خود platform dependent هستند و دیگر وظیفه ی آنهاست که با ساختار سیستم عامل تعامل داشته باشند و کد نوشته را آماده ی اجرا شدن بر ان سیستم عامل بخصوص بکنند. این ابزارها jre و jdk و jvm هستند که هر کدام وظیفه ی بخصوصی دارند که به تفسیر هر کدام از موارد به صورت جدا می پردازیم :

- JDK : همانطور که در بالا هم اشاره شد ، در کد نویسی به زبان Java نیاز به ابزار های مختلفی داریم مانند یک test runner ، debugger ، compiler و ابزارهای از این قبیل که در حین کد نویسی از ابزار هایی هستند که مربوط به java هستند و نیاز به آنها داریم ، در java همه این ابزار ها در یک package جمع شده اند که با نصب ان بر روی سیستم عامل به تمام این ویژگی ها دست پیدا میکنیم به این مجموعه از ابزار ها jdk یا java development kit گفته می شود و در ورژن های جدید تر ان JRE (که خود در درون آن JVM هم هست) بر روی آن موجود است و دیگر نیاز به نصب آن توسط برنامه نویس به صورت جداگانه نیست.
- JRE : این ابزار که مخفف کلمه ی java runtime environment هست ، وظیفه این را به عهده دارد که تمام کارهایی که لازم است از لحظه ی بعد از تولید java bytecode تا لحظه ی اجرای آن را انجام دهد. برای اینکار ابتدا خود JRE تمام java bytecode هایی که نیاز است را به هم link میکند و تمام dependency هایی که نیاز است را inject میکند و بعد به قسمتی دیگر که در ساختار خودش وجود دارد به اسم JVM نتیجه ی حاصل را می سپارد تا کد بر روی سخت افزار اجرا شود.
- JVM : همانطور که در ترم مبانی هم اشاره شد ، هر کدی کی اجرا میشود نیاز به استفاده کردن از سخت افزار را دارد و این کار را سیستم عامل مدیریت میکند . از طرفی ما می دانیم هر سیستم عامل شیوه ی بخصوص خود را دارد برای مدیریت منابع در دسترس دارد ، پس به همین دلیل ما نیاز به ابزاری داریم تا به سیستم عامل با زبانی که می فهمد صحبت کند و منابع مورد نیاز را در وقت نیاز از او دریافت کند یا اگر نرم افزار ما نیاز به دسترسی به یک فایل دارد اجازه ی آن را از سیستم عامل بگیرد و به طور خلاصه اجرای کد ما را به عهده بگیرد ، به این ابزار JVM می گویند که مخفف java virtual machine است.

جواب قسمت ب :

بدیهی است که ابتدا باید کدی به زبان java در فایل های java. نوشته شود.
سپس باید آن را compile کرد به این صورت که هر فایل java. داریم به صورت یک فایل class. یا همان java bytecode در می آید (که به این مرحله compile می گویند)
در مرحله ی بعدی باید java bytecode را که در دست داریم link کنیم به این معنا که اگر از یک dependency استفاده می کنیم آنها inject شوند و از طرفی کدهای دیگری که نوشته ایم خود با هم link شوند و به صورت یک فایل در بیایند و دیگر هیچ قسمتی باقی نمانده باشد که در کد مجهول باشد (به این معنا که تمام method ها و class ها همگی تعریف شان در آن فایل نهایی موجود باشد و بتوان آن را اجرا کرد) و این مراحل در قسمتی از JRE انجام می شود.
حال در مرحله ی بعد باید فایل حاصل از مرحله ی قبل را اجرا کرد ، اما کدی که حالا ما در دست داریم از سیستم عاملی که در آن اجرا میشود هیچ چیزی نمی داند و اصلا به آن وابسته نیست پس JVM که خود قسمتی از معماری JRE است ، وارد می شود و آن را به صورت خط به خط میخواند و معنی آن را میفهمد و کار هایی که سیستم عامل با سخت افزار بکند تا آن دستور اجرا شود را به سیستم عامل انتقال می دهد.
نکته ی مهم این است که در ورژن های بعد تر java مفهوم دیگری به نام JIT هم معرفی شد که تنها مربوط به زبان java نیست اما java یکی از اولین زبان هایی بود که این فلسفه را معرفی کرد و کار هم به این صورت است که : همانطور که گفته شد JVM وظیفه ی این را دارد که کد را به صورت خط به خط بخواند و اجرا کند اما گاها ممکن است که این دستور به صورت بهینه ترین حالت ممکن نباشد و بتوان آن را بهینه تر کرد تا سیستم عامل بتواند سریع تر آن را انجام دهد ، این وظیفه را JIT یا just in time compiler به عهده دارد (البته باید اشاره کرد که JIT کد ما را بهینه نمیکند ، بلکه فایل link شده ی نهایی را بهینه می کند تا JVM بتواند آن را سریع تر اجرا کند و دیگر دست به الگوریتم ما نمی زند)



جواب قسمت ج :

این نوع از داده ها ، داده هایی بسیار ساده هستند که از جمع هیچ داده ی دیگری تشکیل نشده اند و به هیچ وجه نمیتوان آنها را به انواع کوچکتری تجزیه کرد ، در جاوا 8 نوع primitive data موجود است که معروفترین آنها عبارتند از :

- Int : for storing integers(or numbers that don’t have floating point)
- Double : for storing number with floating points
- Char : for storing a single character
- Boolean : a true or false

البته به جز این انواع ، انواع دیگه ای هم موجود هستند که فلسفه ی وجود آنها دادن توانایی مدیریت میزان فضای اشغالی توسط برنامه به برنامه نویس است و میتوان استدلال کرد که به دلیل وضعیت امروزی کامپیوتر ها که حافظه ی کامپیوتر ها دیگر مانند قبل محدود نیست ، قسمت زیادی از کاربرد خودشان را از دست داده امد.

جواب قسمت د :

در طراحی نرم افزار دیدگاه های مختلفی وجود دارد که قسمتی از آنها به paradigm معروف است که معروفترین آنها OOP و Functional هستند و نوعی شاید کمی قدیمی تر Structural است که هر کدام توضیح داده می شوند :

- Functional : در این معماری نرم افزار ، نرم افزار به اجزای کوچکتری تقسیم می شوند که تابع نام دارند و هر کدام وظیفه ی انجام دادن کار بخصوصی را بر عهده دارند. در این نوع معماری تعامل این function هاست که نرم افزار نهایی را می سازد . در این نوع معماری دیگر هیچ روشی موجود برای دسته بندی داده ها موجود نیست و معمولا برنامه نویس خودش به صورت دستی تمام تابع هایی که مربوط به جز مشترکی از نرم افزار هستند را کنار هم قرار میدهد تا بعدا در صورت نیاز بتواند راحت تر debug کند و از طرفی state نرم افزار معمولا به صورت های زیر مدیریت می شود :
  - به صورت متغیر هایی هستند که به صورت global در کل نرم افزار موجود هستند و توابع انها را دستکاری میکنند
  - به صورت متغیر هایی که در تابعی موجود است که تابع در حال اجرا را فراخوانده است و تابع در حال اجرا شدن انها را از parent خود میخواند و یا دستکاری میکند
  - به صورت متغیرهایی که در خود تابع در حال اجرا است و امکان دارد قسمتی از کد همان تابع مقدار آنها خوانده یا تغییر داده شود.
- Structural : در این فلسفه به این دلیل که گاها ممکن است مدیریت کردن state در نوع functional سخت بشود ، به این نوع عمل می کند که ساختاری را تعریف می کند که برنامه نویس می تواند با استفاده از آن متغیر هایی که با هم ربط دارند به هم متصل کند و آنها را تبدیل به واحدی معنادار کند. گاها structure اینگونه تلقی می شوند که آنها نوع جدیدی را به زبان اضافه می کنند . در این نوع معماری معمولا نمی توان هیچ تابعی را به structure اضافه کرد و همین مشکل باعث شد که با ارایه ی OOP دیگر استفاده از زبان های جدیدتر کمتر دیده شود.
- OOP : یا همان object oriented programming که شاید معروفترین فلسفه برای معماری نرم افزار است . در این نوع مفاهیمی که در دنیای واقعیت وجود دارند و برنامه به آنها نیاز دارند به صورت class به دنیای کد معرفی می شوند. در این حالت برای هر مفهومی یک state در نظر گرفته می شود که به صورت متغیر در درون خود class هستند و برخلاف structure ها برای آنها method هایی تعریف می شوند که عملکرد های آن مفهوم را در کد تعریف می کنند. در این نوع از نرم افزار معمولا تعامل بین این class ها است که نرم افزار را به وجود می آورد و از طرفی هم state در این نوع نرم افزار ها به صورت متغیرهایی هستند که در class ها هستند و حال method هایی در همان class یا شاید method هایی در class دیگر آنها را میخوانند یا دستکاری میکنند.

به صورت کلی باید به این مفهوم اشاره کرد در بین نوع OOP , functional معمولا functional برای نرم افزار های متوسط و کوچک که state پیچیده یا اجزای بسیار زیادی دارند پیاده سازی می شود زیرا کمی state management در آنها می تواند پیچیده باشد.

جواب قسمت ه :

- Class : در واقع blueprint هایی هستند که ما در کد خود برای یک مفهوم ارایه میدیم . در این اجزا از زبان ما تمام ویژگی هایی که یک نمونه از این مفهوم باید داشته باشد (از قبیل method ها و property ها) را مشخص میکنیم
- Object : همانطور که در قسمت های قبل توضیح داده شد ، ما در برنامه نویسی مفهوم های دنیا ی واقعی را وارد برنامه نویسی می کنیم این مفهوم ها در واقع همان object ها هستند و class ها پیاده سازی آنها هستند .
- constructor : این method یکی از magic method هایی است که در اکثر زبان های برنامه نویسی که از فلسفه ی oop استفاده می کند موجود است و اولین تابعی است که وقتی نمونه ای از class تولید می شود اجرا می شود و در واقع به نوعی سازنده ی ان class است. به دلیل اینکه تابع constructor اولین تابعی است که در هنگام ساخت نمونه اجرا میشود معمولا از آن برای تعریف کردن یک state برای نرم افزار استفاده می شود و مقدار های مورد نیاز به آن پاس داده می شود تا آنها را در property های class ذخیره کند .
- Method : همانطور که قبلا اشاره شد در oop برای هر مفهوم یک یا چندین عملکرد میتواند تعریف شوند که این عملکرد ها در واقع function هایی هستند که در داخل class هستند و در نرم افزار نقش بخصوصی را باید به عهده داشته باشند.
- Parameter : به صورت کلی به تمام ورودی های یک تابع (چه method باشد چه نباشد) parameter می گوییم
- Instance : یک نمونه است که ما از روی یک class میسازیم . این شی در واقع تمام ویژگی هایی که class را برایش مشخص کرده دارد و علاوه بر آن برای انها مقداری نیز موجود است و از طرفی هر کدام از method های ان را میتوان اجرا کرد.

جواب قسمت و :

همانطور هم که در قسمت های قبل هم توضیح دادم ، در واقع function ها اجزایی از نرم افزارها هستند که به چیز خاصی لزوما وابسته نیستند و معمولا عملکرد خود نرم افزار را تعریف میکند اما method ها جزی از class ها هستند و معمولا عملکردی از ان class را تعریف می کند و بعد عملکرد نرم افزار از مجموعه ای از این عملکرد class هاست که به وجود می اید.