1-Pois em java é impossivel programar sem a utilização das tecnicas orientadas a objetos, é estritamente necessario a criação de classes (que representam objetos “do mundo real”) e que a construção do programa seja voltada a elas, tendo como resultado diversos arquivos e não apenas um arquivo como toda a programação

2-O processo de abstração é trazer objetos do mundo real para a programação, criamos classes que representam tais objetos

3-Classes e metodos que estao presentes em outras classes;

4-um tipo primitivo é um como um tipo ja definido pela linguagem, como o int, float, double

5-um tipo abstrato é um tipo que criamos, como uma struct, porém, em POO criamos classes
6-O garbage colector, como seu nome sugere, vai até a memoria e limpa dados que não possuem referencia no registrador,

7a- primeiro precisamos do JDK, depois de adicionar ao o path o diretorio bin do JDK podemos utilizar o comando “javac” para compilar um arquivo (ou multiplos arquivos caso eles estajam “linkados” na main), exemplo “javac Tstpen.java”
7b-apos executar todos os passos descritos na 7a um arquivo .class sera criado, para rodarmos a aplicação basta utilizar o comando “java”, exemplo “java Tstpen”

8-O bytecode é o arquivo .class gerado apos a compilação do .java, é com ele que rodamos as aplicações e não é possivel alterá-lo diretamente, apenas recompilando o ,java

9-A portabilidade da programação em java se da pela capacidade que ele possui de seu bytecode rodar em diversos sistemas sem que seja necessario sua recompilação. exp:
O programador cria o codigo fonte (.java) e apartir da compilação dele o bytecode, tal bytecode pode rodar em diversos sistemas operacionais sem a necessidade da recompilação do arquivo fonte, o unico requisito necessario é que tal sistema possua um o JRE compativel com seu sistema para que o arquivo .class possa rodar na JVM

10-

11- Acoplagem em uma definição global é o nivel de dependencia entre dois artefatos, podendo ser alta ou baixa. Coesão se refere ao relacionamento que os membros de um modulo possuem, codigos coesos tem seus artefatos fortemente ligados.
Em um conceito de boa pratica de programação uma acoplagem de alto nivel não faz sentido, pois dificultara o acesso, teste e atualização de dados, assim, codigos coesos devem possuir poucas “responsabilidades” facilitando a manutenção e evitando efeitos colaterais

12a- this serve para darmos valores para a variavel “global” (da classe atual), usamos nos setters

12b- super faz uma chamada, apartir da classe filha, a um metodo da classe mãe em uma hierarquia

13a- Setters “public void setNome(String nome){
	this.nome = nome;
}”

13b -public class Filha extends Pai {
	public Filha() {
		super();
		System.out.println("Chamando o construtor da classe-filha...");
	}
	public static void main(String[] args) {
		new Filha();
	}
}

14-A-i-
Encapsulamento possui tres niveis, private: outras classes e aplicações não podem acessar dados com este tipo de encapsulamento;
public: outras classes e aplicações podem acessar e manipular os dados;
protected: apenas classes que herdam metodos/atributos com este tipo de encapsulamento podem acessar os dados e os manipular

14-A-ii-
O encapsulamento feito de maneira coerente as regras de negocio garante mais segurança a aplicação, pois se temos dados que devem(ou não) ser manipulados, no contexto sugerido, podemos restringir (ou não) o acesso a tais metodos e atributos, impedindo o usuario de “estragar” a aplicação

14-A-iii-
Esta afirmação se da devido a capacidade de negarmos o acesso de determinados artefatos a outras classes, impedindo o usuario de os manipular;

14-B-i
Numa hierarquia, a classe mais especializada é a classe filha mais baixa nesta hierarquia (recebe a herança), ela é mais especializada pois geralmente identifica objetos específicos “filhos” da classe mae, (como uma caneta esferográfica).
A classe mais generalizada é aquela da qual podemos criar diversos objetos com suas caracteristas (como a caneta em si)

14-B-ii
Com o sistema de herança podemos criar diversas classes “filhas” as quais irao representar objetos mais especificados do mundo real, podemos ainda definir a classe

mãe como abstract, impedindo a instanciação da mesma e obrigando a criação das classes filhas.

14-B-iii
A reusabilidade vem da capacidade de criarmos varias classes mais especificas com os metodos e atributos da classe mãe, sem termos que reprogramar cada um deles, as classes filhas herdam tais artefatos, gerando um potencial infinito de criação e reutilização, fazendo com que a codificação seja mais dinamica, rapida e eficiente.

14-C
Polimorfismo por sobrecarga acontece quando possuimos dois metodos na mesma classe com os mesmos “nomes” mas com assinaturas/parametros diferentes;

Polimorfismo por sobreescrição ocorre quando possuimos metodos com a mesma assinatura na classe mãe e na classe filha, no caso o da classe filha sera executado;

Polimorfismo por coerção acontece quando temos um objeto com um tipo “X” e com comportamento de outro tipo “Z”

15- arquivo anexado;

16-
É o modo como os objetos se comunicam, funciona baseado em solicitações de uma classe a outra;

17-
O metodo construtor determina quais ações devem ser executadas quanto a criação/instanciação de uma classe, é definido como um metodo cujo nome deve ser o mesmo da classe e sem indicação de retorno, é chamado pelo operador new; exemplo: Copo art = new Artezanal();

18-
Uma classe abstrata é uma classe que não pode ser instanciada, ela existe para ser uma classe mãe na hierarquia de herança;
Um metodo abstract é um metodo de implementação obrigatoria na classe filha (implementado através de sobreescrita);
Uma classe final é uma classe que não passa herança para frente, não pode ter classes filhas;
Um atributo final é um atributo constante, apos sua declaração e atribuição não é possivel mudar seu valor;
Um metodo final é um metodo que não pode ser sobreescrito por uma classe filha;

19-

Interface em java define ações que devem ser obrigatoriamente executadas, mas que cada classe pode executar de maneira diferente