```java
package heranca;

class Mamifero {
	public int a;
	protected int b;
	int c;
	private int d;

	Mamifero(int w, int x, int y, int z) {
		a = w;
		b = x;
		c = y;
		d = z;
	}

	void imprime() {
		System.out.println("Metodo imprime da super-class Mamifero.");
		System.out.println("A = " + a);
		System.out.println("B = " + b);
		System.out.println("C = " + c);
		System.out.println("D = " + d);
	}

	public int getA() {
		return a;
	}

	protected int getB() {
		return b;
	}

	int getC() {
		return c;
	}

	private int getD() {
		return d;
	}

}
```

```java
package heranca;

class Cachorro extends Mamifero{

	int e;

	public Cachorro(int w, int x, int y, int z, int e) {
		super(w, x, y, z);
		this.e = e;
	}

	int getE(){
		return e;
	}

	void acessaCampos() {
		System.out.println("Metodo acessaCampos da sub-classe Cachorro.");
		System.out.println("A = " + getA());
		System.out.println("B = " + getB());
		System.out.println("C = " + getC());
		// System.out.println("D = " + getD()); // NAO FUNCIONA pois getD() eh
private
		System.out.println("E = " + getE());
	}

	void imprime(){
		System.out.println("Metodo imprime da sub-classe Cachorro.");
		super.imprime();
		System.out.println("E = " + getE());
	}

	void late(){
		System.out.println("Metodo late da sub-classe Cachorro.");
	}

}
```

```java
package heranca;

public class Exemplo1 {
	public static void main(String[] args)
	{
		Mamifero e1 = new Mamifero(1,2,3,4);
		Cachorro e2 = new Cachorro(6,7,8,9,10);
		Mamifero e3 = new Cachorro(11,12,13,14,15); // Funciona pois toda sub-classe "também é" classe
		// Cachorro e4 = new Mamifero(21,21,22,23); // Nao funciona pois uma super-classe não é classe
		e1.imprime();
		e2.imprime();
		e3.imprime();
		((Mamifero)e3).imprime();
		// e3.late(); // Nao funciona pois, mesmo e3 contendo um objeto do tipo Cachorro, a variável é do tipo Mamifero
		((Cachorro)e3).late(); // o 'cast' permite a execucao do metodo late, note que o programador precisa ter certeza que o conteudo de e3 é compatível com Cachorro
		// ((Cachorro)e1).late(); // Nao funciona pois o objeto que está em e1 não é compatível com o tipo Cachorro [porém não dará erro de compilação e sim de execução]

/* Note que:
 * 1. e3 eh uma variavel do tipo Mamifero e recebe um objeto do
 *    tipo Cachorro;
 * 2. a classe Cachorro nao tem acesso (visibilidade) ao atributo
 *    d da super-classe, porem este atributo existe e eh impresso.
 * 3. o cast da penultima linha significa "leia e3 como sendo do tipo Cachorro, garantindo
 *    assim acesso ao metodo 'late()' da sub-classe Cachorro.
 *    Isto funciona pois a instancia e3 contem um objeto do tipo Cachorro
*/
	}
}

/* RESULTADO DA EXECUCAO DO PROGRAMA:
Metodo imprime da super-class
Mamifero.
A = 1
B = 2
C = 3
D = 4
Metodo imprime da sub-classe Cachorro.
Metodo imprime da super-class
Mamifero.
A = 6
B = 7
C = 8
D = 9
E = 10
Metodo imprime da sub-classe Cachorro.
Metodo imprime da super-class
Mamifero.
A = 11
B = 12
C = 13
D = 14
E = 15
Metodo imprime da sub-classe Cachorro.
Metodo imprime da super-class
Mamifero.
A = 11
B = 12
C = 13
D = 14
E = 15
*/
```

```java
package heranca;

class Pessoa {
	String nome;
	int RG;
	Pessoa(String nome, int RG){
		this.nome = nome;
		this.RG = RG;
	}

	void respirar(){
		System.out.println("Pessoa respirando.");
	}

	void dormir(){
		System.out.println("Pessoa dormindo.");
	}

}
```

```java
package heranca;

public class Digitador extends Pessoa{
	int letrasPorSegundo;
	Digitador(int letras, String nome, int rg){
		super(nome,rg);
		letrasPorSegundo = letras;
	}

	int numeroDeLetrasPorSegundo(){
		return letrasPorSegundo;
	}
}
```

```java
package heranca;

// A classe Aluno é abstrata (não pode ser instanciada)
abstract class Aluno extends Pessoa {
	String nome;
	int RG;
	int numeroUSP;

	/* construtores não podem ser final */
	Aluno(String n, int rg, int nUSP){
		super(n,rg);
		numeroUSP = nUSP;
	}

	void respirar(){
		System.out.println("Aluno respirando.");
	}

	final void fazerTrabalhos(){
		System.out.println("Trabalhando.");
	}


	// Método abstrato, precisa ser implementado nas sub-classes não abstratas
	// um método não pode ser abstrato e estático ao mesmo tempo
	abstract void estudar();

}
```

```java
package heranca;

public class AlunoRegular extends Aluno{
	/* Já que aluno não tem construtor sem parâmetros é necessário criar um
construtor aqui */
	AlunoRegular(String nome, int RG, int NUSP){
		super(nome,RG,NUSP);
	}

	void estudarMuito(){
		System.out.println("Estudando muito!");
	}

	// PRECISA implementar todos os métodos abstratos da super-classe
	void estudar(){
		System.out.println("Aluno Regular Estudando");
	}


}
```

```java
package heranca;
/* a classe AlunoEspecial nao pode estender Aluno e Digitador pois, em Java,
 * cada classe só pode extender uma classe.  */
public final class AlunoEspecial extends Aluno{
	/* Já que aluno não tem construtor sem parâmetros é necessário criar um construtor aqui */
	AlunoEspecial(String nome, int RG, int NUSP){
		super(nome,RG,NUSP);
	}

	void respirar(){
		super.respirar(); // NAO é possível usar, super.super.respirar (super.super não é permitido)
		System.out.println("Aluno especial respirando.");
	}

	/*  Este metodo não pode ser implementado aqui por ser final na super-clase (Aluno)
	void fazerTrabalhos(){
		...
	}   */


	// PRECISA implementar todos os métodos abstratos da super-classe
	void estudar(){
		System.out.println("Aluno Especial Estudando");
	}

}
```

```java
package heranca;

public class AlunoExecutar {
	public static void main(String[] args) {
		Pessoa pessoas[] = new Pessoa[3];
		pessoas[0] = new Pessoa("José",12345);
		// A classe aluno não pode ser instanciada por ser abstrata
		pessoas[1] = new AlunoEspecial("Maria",45678,123);
		pessoas[2] = new AlunoRegular("João",88990,222);

		pessoas[0].respirar();
		System.out.println();
		pessoas[1].respirar();
		System.out.println();
		pessoas[2].respirar();
		System.out.println();

		((Aluno)pessoas[1]).estudar();

		Aluno a1 = new AlunoEspecial("Julio",33333,333);
		Aluno a2 = new AlunoRegular("Carla",44444,444);

		a1.estudar();
		a2.estudar();

		System.out.println();

		// ((AlunoRegular)a1).estudarMuito();  // NAO FUNCIONA (compila e gera exceção) pois o cast é incorreto (um AlunoEspecial não é um AlunoRegular)
		((AlunoRegular)a2).estudarMuito();
		// (AlunoRegular)a2.estudarMuito();  // NAO FUNCIONA (não compila) pois tenta fazer o cast após a execução

	}
}
```