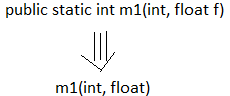
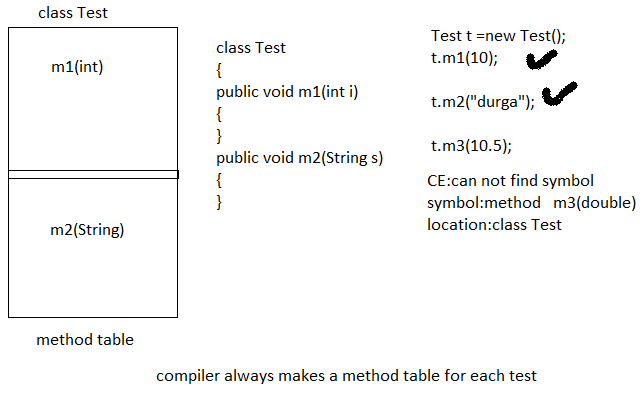
Method Signature:

In java method signature consist of methods name followed by argument types  
   
note: method type is not a part of method signature in java

1.Compiler will use method signature to resolve method calls  
example:  
 

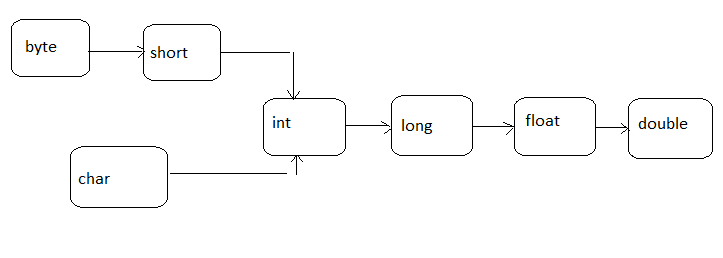
2.Within a class two method with same signature is not allow  
example:

Class Test{

Public void m1(int i)=>m1(int)  
{

}

Public void m1(int j)=>m1(int)  
{

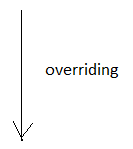
}  
 CE: m1(int) is already defined test  
  
  
 Overloading  
  
two methods are said to be overloaded if and only is both methods having name but different argument types.  
  
example:  
abs(int i);  
abs(string s);  
abs(double d);  
  
method overloading reduce the complexity of programming   
  
Some deep knowledge in method overloading:  
  
Case:1  
Automatic promotion in overloading: while resolving overloaded method if exact match method id not available then we don’t get any compile time error immediately first it will promote argument to the next level and check whether matched method is available or not if match method is available then It will be consider if the match method is not available then compiler promote argument once again to the next level this process will be continued until all possible promotions still if the match method is not available then we will get compile time error.  


Code example:  
class Test  
{  
public void m1(int i)

{  
sopln(“int arg”);  
}  
public void m1(float f)  
{  
sopln(“float arg”);  
}  
public static void main(String[] args)  
{  
Tets t= new Test();  
t.m1(10);---------🡪int arg  
t.m1(10.5f);----🡪float arg  
t.m1(‘a’);-------🡪int arg  
t.m1(10L);-----🡪float arg  
t.m1(10.5);----🡪CE:can not find symbol   
 symbol: method m1(double)  
 location: class Test  
  
Case:2  
  
class Test  
{  
public void m1(String s)  
{  
sopln(“string version”)  
}  
public void m1(Object o)  
{  
sopln(“Object version”)  
}  
 public static void main(String[] args)  
{  
Test t = new Test();-------------------🡪Object version  
t.m1(new Object());------------------🡪String version  
t.m1(“java”);  
t.m1(null);  
}  
}  
  
Note:  
 Object<-------------String  
while resolving overloaded method compiler will always give the precedence to child class object then compare to parent type object  
  
  
Case 3:  
class Test  
{  
public void m1(int I, float f)  
{  
sopln(“int – float version”);  
}  
public void m1(float f, int i)  
{  
sopln(“float – int version“);  
}  
Public static void main(String [] args)  
{  
Test t= new Test();  
t.m1(10,10.5f);-------------🡪int-float version  
t.m1(10.5f,10);------------🡪 float-int version

t.m1(10,10);---------------🡪CE: reference to m1() ambiguous  
t.m1(10.5f,10.5f):---------🡪CE: can not find the symbol  
 symbol: method(float,float)  
 location : class Test  
}  
}

Case:4  
  
special condition  
class Animal  
{  
class Monkey extends Animal  
{  
}  
class Test  
{  
public void m1(Animal a);  
{  
sopln(“Animal version”);  
}  
public void m1(Monkey m)  
{  
sopln(“Monkey version”);  
}  
public static void main(Strings[] args)  
{  
Test t = new Test();  
Animal a = new Animal();  
t.m1(a)---------------🡪Animal version  
Monkey m = new Monkey();  
t.m1(m)--------------🡪Monkey version  
Animal a1 =new Monkey();  
t.m1(a1)-------------🡪Animal version  
}  
}

Note: In overloading method resolution always takes care by compiler based on reference type.  
In overloading run time object won’t play any role.  
  
  
  
  
  
  **Overriding**  
Whatever methods parent have by default available to the child through inheritance if child class not satisfy with parent class implementation then child is allow to redefine that method based his requirement this process is called overriding.  
the parent class method which is overridden is called overridden method and child class method which is overriding is called overriding methods.  
example:  
 class Parent  
{  
public void property()  
{  
sop(“land+cash+gold”);  
}  
public void marry()---------------------------🡪overridden method  
{  
sop(“jaya”);  
}   
}  
class Child extends Parent  
{  
public void marry()---------------------------🡪overriding method  
{  
sop(“xyz”);  
}  
}  
  
example:2  
  
class Test  
{  
public static void main(Strings [] args)  
{  
Parent p = new Parent();  
p.marry();-----------------------------------🡪Parent method  
  
Child c= new Child();  
c.marry();------------------------------------🡪child method  
  
Parent p1 = new Child();  
p1.marry();---------------------------------🡪child method  
  
}  
}

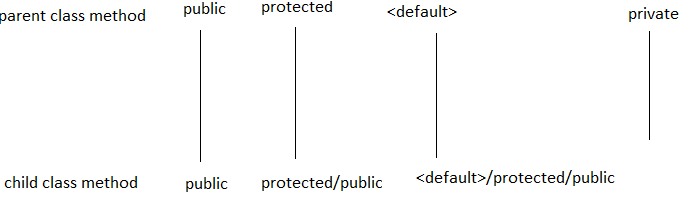
Note:   
In overriding method resolution always takes care by JVM based on run time object and hence overriding is also consider as run time polymorphism or dynamic polymorphism or late binding  
  
Rules for overriding:  
1. In overriding method names and arguments type must be match i.e. method signature must be same.  
2. In overriding return type must be same but this rule is applicable until 1.4 version only for 1.5 version onwards   
we can take co- variant return types.  
 according to this child class method return type did not be same as parent method return type.  
example:  
 class Parent {  
public Object m1()  
{  
return null;  
}  
}  
class child extends Parent  
{  
public String m1()  
{  
return null;  
}  
}  
\*\*\*\* this is applicable after 1.5 version>

Note:

1.Co variant return type applicable only for object type but not for primitive type.  
2.parent class private method is not available to child and hence overriding concept not applicable for private method.  
3.private methods can not be override.  
 example:

class p   
{  
private void m1()  
{  
}  
}  
class c extends p   
{  
private void m1()  
{  
}  
}

this is valid but not override method, its look like override method  
3.we cant override parent class final method in child classes if we are trying to override we will get compile time error.  
  
example:  
 class p   
{  
public final void m1  
{  
public final void m1()  
{  
}  
}  
class c extends p  
{  
public void m1()  
{  
}  
}  
  
  
CE: m1() in c can not be override m1() in p;  
overridden method is final.  
  
Case:

Parent class abstract methods we should override in child class to provide implementation  
example:  
abstract class p  
{  
public abstract void m1();  
}  
class c extends p  
public void m1()  
{  
}  
}  
  
Point to be remember:  
while overriding we can not reduce scope of access modifier but we can increase the scope of modifier.  
  


Overriding concept is not applicable for private modifier.