OVERLOADING

Two methods are said to be overloaded if and only if both methods having same name but different argument types.

abs(int i)

abs(long l)

abs(float f)

**public void** m1() {}

**public void** m1(**int** i) {}

**public void** m1(**int** i, **int** ii) {}

**public void** m1(**long** l) {}

It is also know as **compile time polymorphism or static polymorphism or early binding** because method resolution always takes care by **compiler** based on reference type.

**Auto Promotion in Overloading:**

While resolving overloaded methods, if exact matched method is not available then we wont get any compile time error immediately. First it will promote argument to the next level and check whether matched is available or not. If matched method is available then it will be considered.

If the matched method is not available then compiler promotes argument once again to next level, this process will be continued until all possible promotion. Still if the matched method is not available we will get compile time error. The following are all possible promotions in overloading

**byte--> short -->**

**|**

**-- int --> long --> float --> double**

**|**

char -->

This process is called automatic promotion in overloading.

Example:

**class** Test {

**public void** m1(**int** i){}

**public void** m1(**float** f){}

}

Test t = **new** Test();

t.m1(1);

t.m1(**'a'**);

t.m1(1.1f);

t.m1(10l);

*//t.m1(10.5); not possible because it is double*

Case 2:

While resolving overloaded methods compiler will always gives the precedence for child class argument when compared with parent type argument.

**class** Test {

**public void** m1(String str){}

**public void** m1(Object obj){}

}

Test t = **new** Test();

t.m1(**null**); *//will call string, because work can be completed by child object itself*

*//so no need to invoke parent object*

t.m1(**"String"**);

t.m1(**new** Object());

Case 3:

**class** Test {

**public void** m1(String str){}

**public void** m1(StringBuffer stringBuffer){}

}

Test t = **new** Test();

*//t.m1(null); //Ambigious call, because both String and StringBuffer are at same level :(*

t.m1(**"String"**);

t.m1(**new** StringBuffer());

Case 4:

**class** Test {

**public void** m1(**int** i, **float** f){}

**public void** m1(**float** f, **int** i){}

}

Test t = **new** Test();

t.m1(10, 10.5f);

t.m1(10.5f, 10);

*//t.m1(10, 10); ambigious*

Case 5:

**class** Test {

**public void** m1(**int** i){}

**public void** m1(**int**... i){}

}

Test t = **new** Test();

t.m1(); *// var args - zero parameter*

t.m1(10, 22); *// var args*

t.m1(20);*//general method will get chance, backward compatability*

Case 6:

In overloading method resolution will be take care by reference type. In overloading runtime object wont play any role.

**class** Test {

**public void** m1(Animal animal){}

**public void** m2(Monkey monkey){}

}

**class** Animal {}

**class** Monkey **extends** Animal {}

Test t = **new** Test();

t.m1(**new** Animal());

t.m1(**new** Monkey());

Animal a = **new** Monkey();

t.m1(a); *//will call animal*