COMP 250

Lecture 33

type conversion
polymorphism (intro only)
Class class

Nov. 24, 2017
Primitive Type Conversion

double
float
long
int
short
char
byte
boolean

non-integers

integers

In COMP 273, you will learn details of how number representations are related to each other.

But you should have some intuitive ideas....

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
# Primitive Type Conversion

<table>
<thead>
<tr>
<th>Type</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>char</td>
<td>2</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
</tr>
</tbody>
</table>

Here, wider usually *(but not always)* means more bytes.

- narrower
- wider
Examples

```c
int i = 3;
double d = 4.2;
d = i ;
  // widening
```
Examples

int i = 3;
double d = 4.2;
    d = i; // widening

d = 5.3 * i; // widening (by "promotion")
    i = (int) d; // narrowing (by casting)
float f = (float) d; // narrowing (by casting)

For primitive types, both widening and narrowing change the bit representation. (See COMP 273.)

For narrowing conversions, you get a compiler error if you don’t cast.
Examples

```java
int i = 3;
double d = 4.2;
d = i;  // widening

d = 5.3 * i;  // widening (by "promotion")
i = (int) d;  // narrowing (by casting)

float f = (float) d;  // narrowing (by casting)

char c = 'g';

int index = c;  // widening

char c = (char) index;  // narrowing
```
Although a subclass is narrower, it has more fields and methods than the superclass (in that it inherits all fields and methods from superclass).
class Dog
String serialNumber
Person owner
void bark():

extends
class Beagle
void hunt():

extends
class Poodle
void show():

extends
class Doberman
void fight():

Dog myDog = new Beagle(); // upcast, widening

This is similar to:

double myDouble = 3; // from int to double.
Dog    myDog = new Beagle();       // Upcast, widen.

Poodle myPoodle = myDog;

myDog.show()
Dog    myDog = new Beagle();   // Upcast, widen.

Poodle myPoodle = myDog;       // Compiler error.

// Implicit downcast Dog to Poodle is not allowed.

myDog.show();      // Compiler error.

// Poodle has show() method,
// but Dog does not.
Dog myDog = new Beagle(); // Upcasting.

Poodle myPoodle = (Poodle) myDog; // Downcast
// Narrowing

myPoodle.show()

((Poodle) myDog).show()
Dog   myDog = new Beagle();     // Upcasting.

Poodle  myPoodle = (Poodle) myDog;

    // allowed by compiler

    myPoodle.show()     // allowed by compiler
    // but runtime error: Dog object
    // does not have show() method

    ((Poodle) myDog).show()  

    // allowed by compiler, but runtime error for same reason
Most of examples above concerned compile time issues.

We next examine runtime issues.
COMP 250

Lecture 33

type conversion
polymorphism (intro only)

Class class

Nov. 24, 2017
Recall example from lecture 30

```java
class Dog
    String serialNumber
    Person owner
    void bark()
    {
        print "woof"
    }

class Beagle
    void hunt()
    void bark()
    {
        print "aowwwuuu"
    }

class Doberman
    void fight()
    void bark()
    {
        print "Arh! Arh! Arh!"
    }

Dog myDog = new Beagle();
myDog.bark();

→ ??????? (which bark?)
```
Recall example from lecture 30

class Dog
String serialNumber
Person owner
void bark()
{
print "woof"
}

extends
class Beagle
void hunt()
void bark()
{
print "aowwwuuuu"
}

Dog myDog = new Beagle();
myDog.bark();

→ "aowwwuuuu"

extends
class Doberman
void fight()
void bark()
{
print "Arh! Arh! Arh!"
}
Polymorphism

“poly” = multiple
“morph” = form

We have seen the idea already:
The object type (run time) can be *the same or narrower* than the declared type (compile time).

More general discussion about polymorphism in higher level courses e.g. COMP 302.
Polymorphism
(the following is an important idea, not a formal definition)

Compile time:
Suppose a reference variable has a declared type:

\[ C \quad \text{varC} \quad \text{// C is a class} \]
\[ A \quad \text{varA} \quad \text{// A is an abstract class} \]
\[ I \quad \text{varI} \quad \text{// I is an interface} \]

Runtime:
\[ \text{varC} \quad \text{can reference any object of class C or any object of a class that extends C.} \]
\[ \text{varA} \quad \text{can reference any object whose class extends abstract class A.} \]
\[ \text{varI} \quad \text{can reference any object whose class implements interface I.} \]
boolean b;
Object obj;

if ( b )
    obj = new Cat();
else
    obj = new Dog();

System.out.print( obj );

// Which toString() method that gets called
// depends on the object referenced by obj.
How does (runtime) polymorphism work?

To answer this question, I first need to explain how classes are represented in a running program.
COMP 250

Lecture 33

type conversion
polymorphism (intro only)

Class class

Nov. 24, 2017
Java code
(.java text file)

compiler

.class file
Java .class file ("byte code")

It has a specific format for information such as:

- the class name
- fields (names, types)
- methods (signature, return type, instructions)
- superclass
- ....

https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html
Example

Dog.java

text file

compiler

Dog.class

class file

runtime

The class is “loaded” into the JVM

What is this?
Dog.java

text file

compiler

dog.class

class file

Runtime

The class is "loaded" into the JVM.

Dog

"class descriptor"
The term “class descriptor” is not standard. So don’t look it up.

It is an object that contains all the information about a class.

If it is an object, then what class is it an instance of?

```
Dog
  class descriptor

String
  class descriptor

Beagle
  class descriptor

LinkedList
  class descriptor
```
A “class descriptor” is an instance of the Class class.

It has many methods:

```java
class Class

Class   getSuperClass()
Method[ ] getMethods( )
Field[ ]  getFields( )
String   getName( )
```

A Dog object is an instance of the Dog class.

A String object is an instance of the String class.

An Object object is an instance of the Object class.

A Class object ("class descriptor" object) is an instance of the Class class.
Each class descriptor is an instance of the Class class.

This figure shows objects in a running Java program.
This figure shows classes in the Java class hierarchy.

```java
class Object {
    boolean equals(Object);
    int hashCode();
    String toString();
    Object clone();
    Class getClass();
}

class Animal :

class Dog : Animal {
}

class Beagle : Dog {
}
```

This figure shows classes in the Java class hierarchy.
All classes inherit the `Object.getClass()` method which returns the class descriptor for that object.

This figure shows objects in a running Java program.
All classes inherit the `Object.getClass()` method, which returns the class descriptor for that object.

This figure shows objects in a running Java program.
This figure shows classes in the Java class hierarchy.

class Object

- boolean equals( Object )
- int hashCode()
- String toString()
- Object clone()
- Class getClass()

extends Class

class Animal

extends Class

class Dog

extends Class

class Beagle

extends (automatic)

class Class

- Class getSuperClass()
- Method[ ] getMethods()
- Field[ ] getFields()
- String getName()
This figure shows objects in a running Java program.

The `getSuperClass()` method cannot be invoked by the objects above. Why not?
This figure shows objects in a running Java program.
We’ll see more about how this works next week...