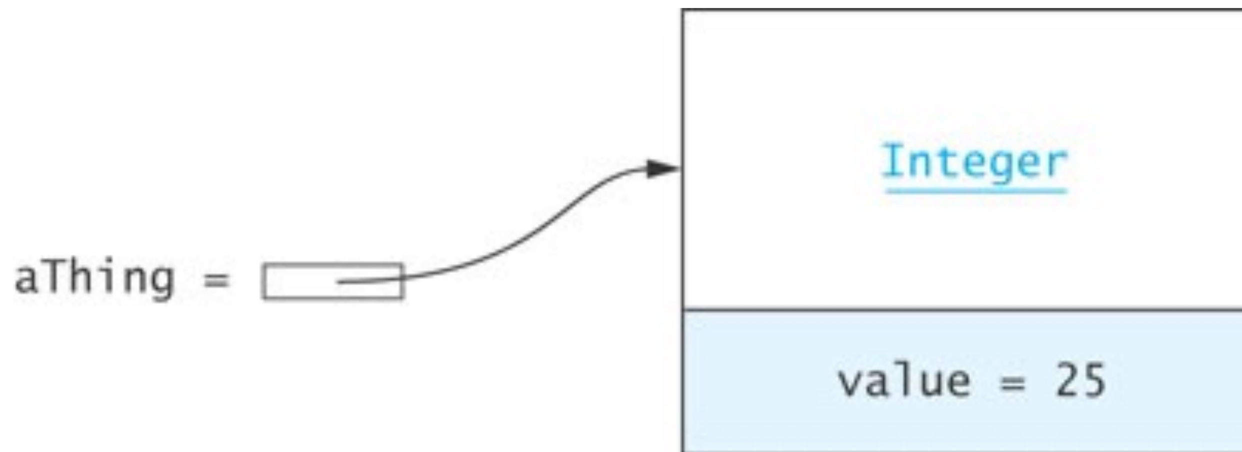
INTEGER example

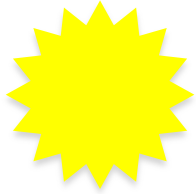
Compile time Operations Determined by Type of Reference Variable

- As shown previously with Computer and Notebook, a variable can refer to an object whose type is a subclass of the variable's declared type
- Java is strongly typed

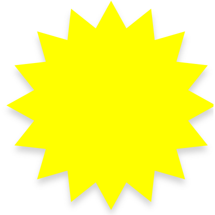
```
Object aThing = new Integer(25);
```

  - The compiler always verifies that a variable's type is a superclass of the class of every expression assigned to the variable (e.g., class Object must include class Integer)





Operations Determined by Type of Reference Variable (cont.)



- The type of the variable determines what operations are legal

```
Object aThing = new Integer(25);
```

- The following is legal:

```
aThing.toString();
```

- But this is not legal:

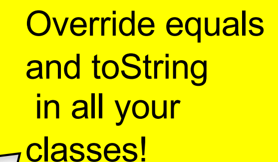
```
aThing.intValue();
```

- Object has a toString() method, but it does not have an intValue() method (even though Integer does, the reference is considered of type Object)

Class Object

Single inheritance in Java, Object is the root

- Object is the root of the class hierarchy
- Every class has Object as a superclass
- All classes inherit the methods of Object but may override them
- Some methods of Object



Method	Behavior
`boolean equals(Object obj)`	Compares this object to its argument.
`int hashCode()`	Returns an integer hash code value for this object.
`String toString()`	Returns a string that textually represents the object.
`Class<?> getClass()`	Returns a unique object that identifies the class of this object.

Operations Determined by Type of Reference Variable (cont.)

- The following method will compile,

```
aThing.equals(new Integer("25"));
```

- Object has an equals method, and so does Integer
- Which one is called? Why?

- Why does the following generate a syntax error?

```
Integer aNum = aThing;
```

- Incompatible types!

Casting in a Class Hierarchy

- Casting obtains a reference of a different, but matching, type
- **Casting does not change the object!**
  - It creates an anonymous reference to the object
  - It tells the compiler, “I know what I am doing” (even if you don’t really know)

```
Integer aNum = (Integer) aThing;
```

- Downcast:
  - Cast superclass type to subclass type
  - **Java checks at run time to make sure it’s legal**
  - If it’s not legal, it throws ClassCastException

Downcasting Example -Visual

- Integer aNum = (Integer) aThing;