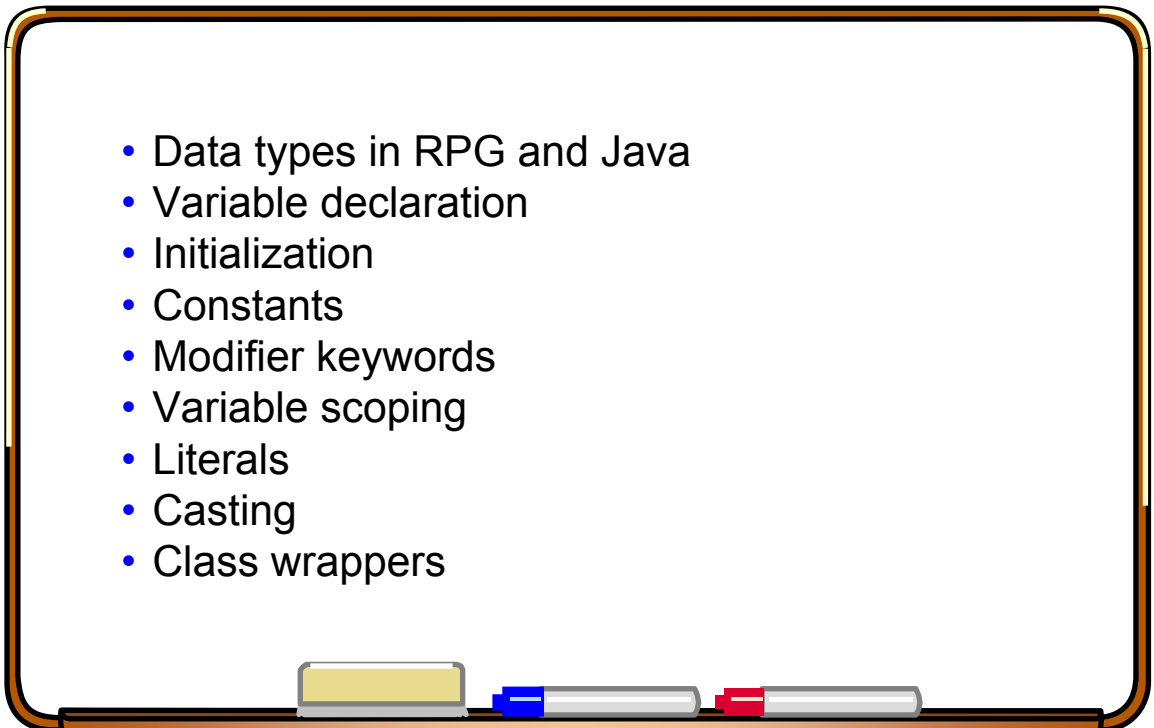
Data types

Topics



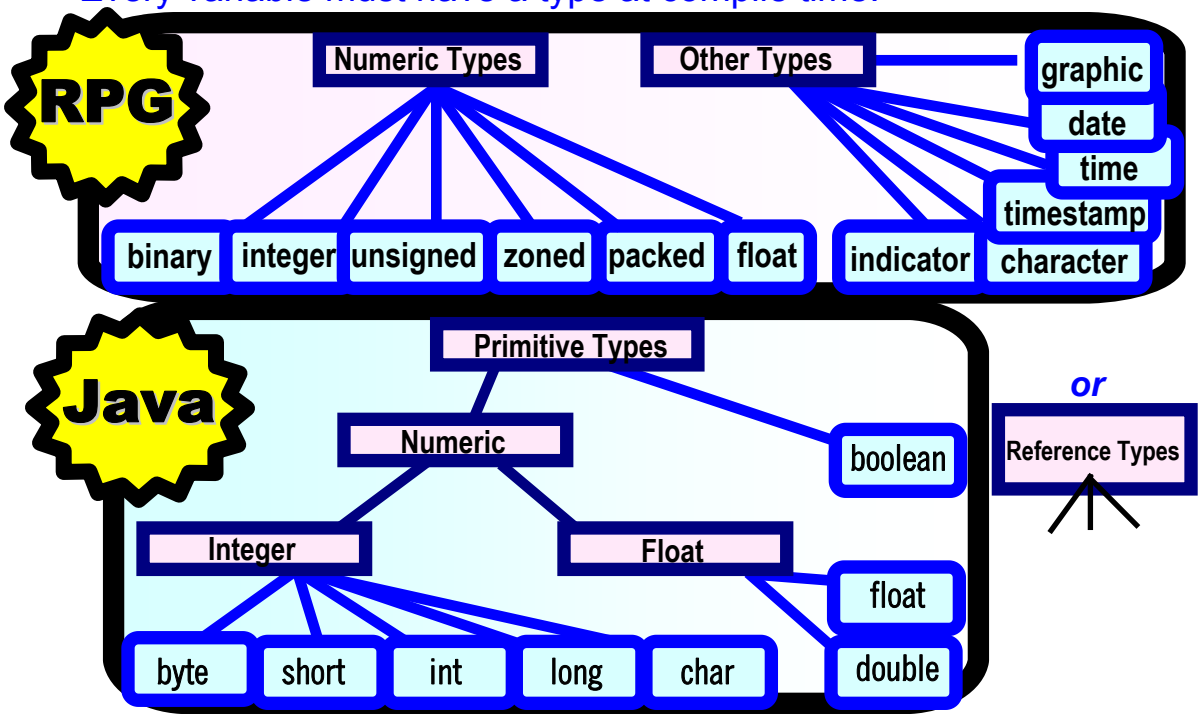
Unit objectives

After completing this unit, you should be able to:

- Describe how Java declares and represents program data
- Use Java syntax to declare and initialize program variables to store data
- Distinguish variables of different access scope
- Use various classes in the JDK to manipulate data

Data type overview

- RPG and Java are strongly typed languages:
  - Every variable must have a type at compile time.

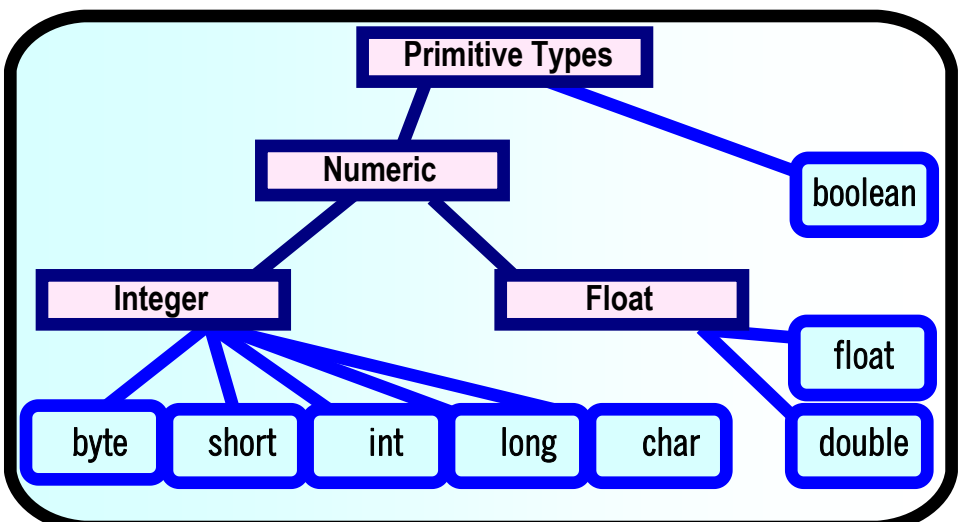


Data types in RPG

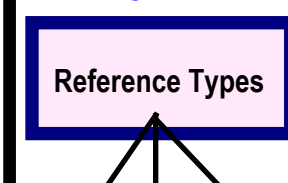
Data Type	Description
Numeric	Can be binary, zoned, packed, integer, unsigned or float
Character	One or more characters
Unicode	One or more UNICODE characters (>=V4R4)
Graphic	One or more double byte (DBCS) characters
Date	Date data type
Time	Time data type
TimeStamp	Consists of Date and Time combined
Indicator	Can contain a '0' or '1' (>=V4R2)
Basing Pointer	Contains a memory address
Procedure Pointer	Contain an address to a procedure

Java data types

- Primitive types: Similar to RPG's data types
  - Predefined in the language
  - Built-in support: operators and conversion rules
- Reference types: Variables of a class type

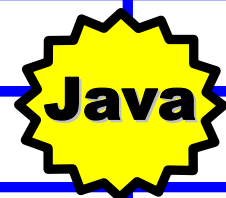


or



Java primitive types

Data Type	Example	Description
Integer	`int i;`	*4 byte signed: about +- 2 billion*
Long	`long l;`	*8 byte signed: about +- huge #*
Byte	`byte b;`	*1 byte signed: -128 to + 127*
Short	`short s;`	*2 byte signed: -32768 to 32767*
Character	`char c;`	*2 byte unicode: only 1 character!*
Boolean	`boolean flag;`	**true** *or* **false**
Float Single	`float f;`	*32 bit*
Float Double	`double d;`	*64 bit*



RPG versus Java data types

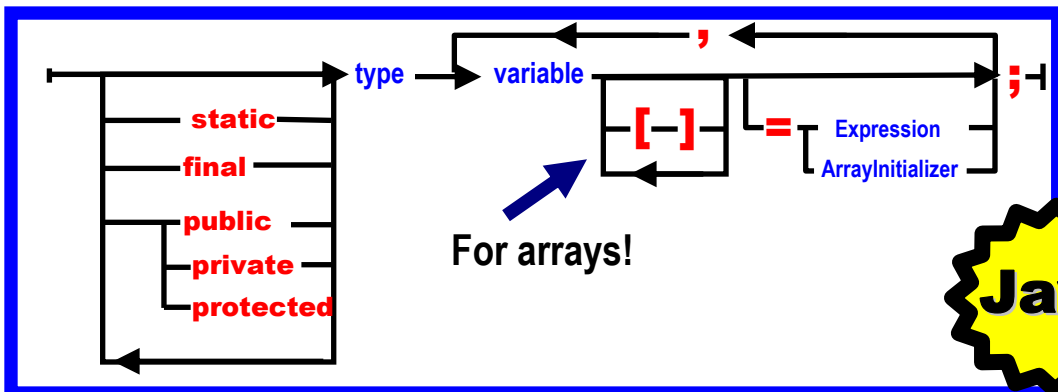
RPG	Java	Comments
numeric (no decimals)	byte or short or int or long or BigInteger class	Depends on length. RPG now supports 1,2,4,8 byte integers
numeric (with decimals)	float or double, or BigDecimal class	Depends on length. BigDecimal is a Java supplied class, not a primitive
float (length 4)	float	Both are IEEE standard
float (length 8)	double	Both are IEEE standard
basing pointer	object reference	Both are memory addresses, but Java does not permit address math
procedure pointer	not available	
character (length one)	char	Single character only
character (length n)	String class	Covered in later chapter
graphic	String class	Covered in later chapter
indicator	boolean	'1' = true, '0' = false
date	java.util.Date class	See Date and Time chapter
time	java.util.Time class	See Date and Time chapter
timestamp	java.util.Date class	See Date and Time chapter

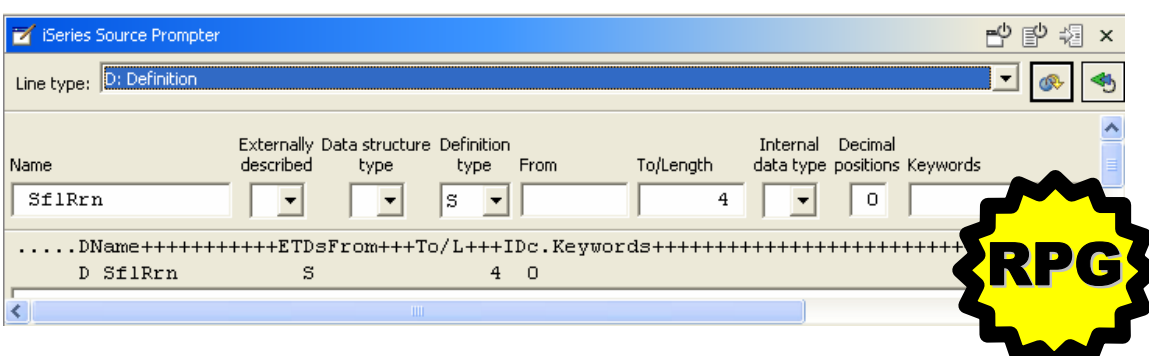
Declaring in RPG

```
+*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+...
************ Beginning of data *********************************
  FQSYSPRT   O    F   80        PRINTER OFLIND(*INOV)
  D FIRST           S                7A   INZ('George ')
  D*-------------------------------------------------------------
  C     *LIKE       DEFINE     FIRST          LAST              -3
  C                 MOVE       ' AGE WAS--->  AGETEXT           12
  C                 MOVE       *ON            *INLR
  C*-------------------------------------------------------------
  OQSYSPRT   E            RESULT
  O                       FIRST                5
  O                       LAST                10
  O                       AGETEXT             22
  O                       AGE                 26
```

- On the C specification
- Using the Define operation code
- On the new Definition specification

Declaration syntax

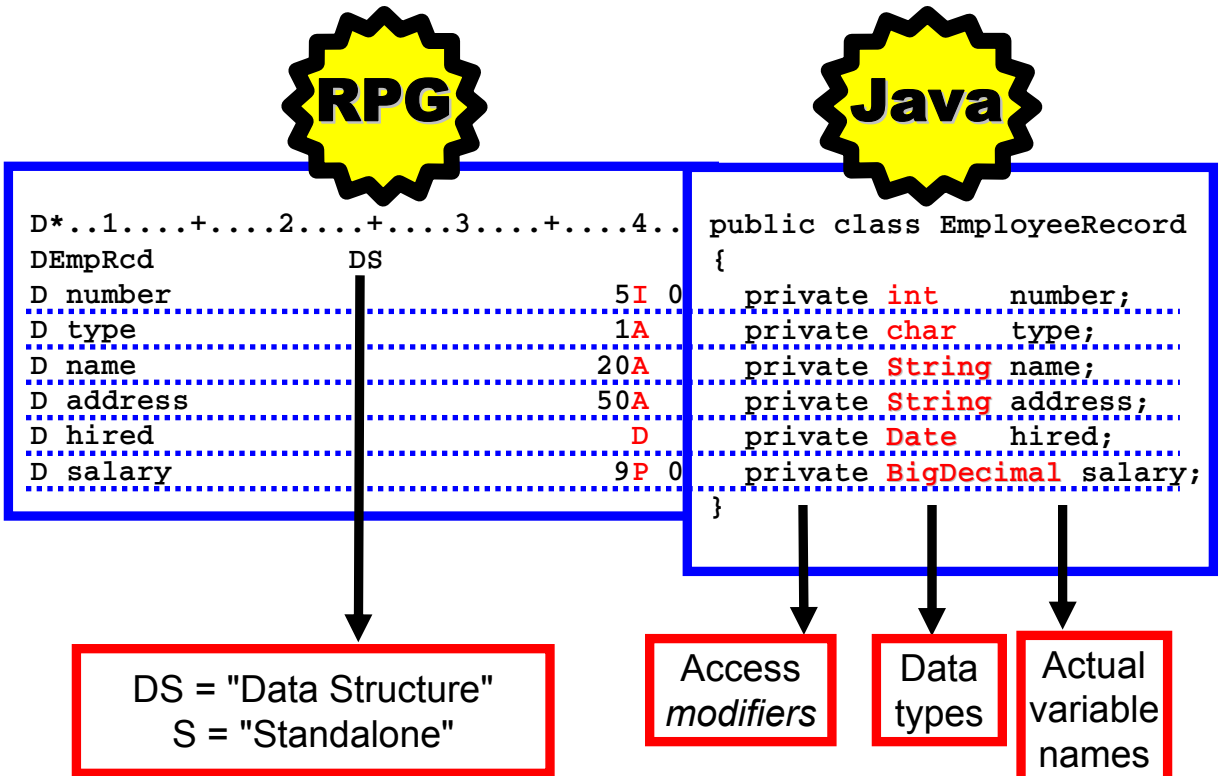




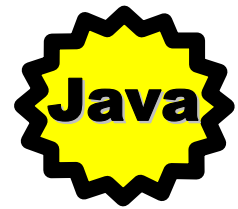
Data type field

Data Type	Description
A	Character. Specify the VARYING keyword for a variable length field.
B	Numeric (Binary format)
C	Unicode string
D	Date
F	Numeric (Float format)
G	Graphic (Fixed or variable-length format) . Specify the VARYING keyword for a variable length field.
I	Numeric (Integer format) : length (in digits)=3,5,10,20
N	Character (Indicator format)
P	Numeric (Packed decimal format)
S	Numeric (Zoned format)
T	Time
U	Numeric (Unsigned format) : length (in digits) = 3, 5, 10, 20
Z	Timestamp
*	Basing pointer or procedure pointer

Declaring variables



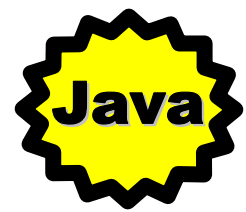
Where is the length?



- You do not specify number of digits!
  - Data type determines number of bytes
    - Which determines how much variable can hold
    - For example, short holds -32768 to 32767
- Usually you will use:
  - Integer ("int") when no decimals (unless numbers > 2 billion)
  - BigDecimal class when decimals needed
  - String class when dealing with characters

What about editing?

- *No* editing is allowed on primitive types in Java!
  - Only numeric data allowed in byte, short, int, long, float or double
  - Any UNICODE character in char
  - Only true or false in Boolean
- How do you edit numeric data for output?
  - System.out.println
    - Formats numeric primitive variable data for you
    - Formats objects for you by calling their toString method
  - DecimalFormat class in java.text package
    - For defining patterns that can be applied to data



Declaring and initializing

```
public class EmployeeRecord
{
    private int     number  = 0;
    private char    type    = 'R';
    private String  name    = "Joe Public";
    private String  address = "1 Young St";
    private Date    hired   = new Date();
    private BigDecimal salary  = new BigDecimal("30000.00");
}
```

Java

Note: Use **new** operator because type is a class, not a primitive.

RPG

```
D*..1....+....2....+....3....+....4....+....5
DEmpRcd           DS
D number                         5I 0 INZ(0)
D type                           1A   INZ('R')
D name                          20A   INZ('Joe Public')
D address                       50A   INZ('1 Young St')
D hired                           D   INZ(D'1999-12-31')
D salary                         9P 0 INZ(30000)
```

Declaring constants

"**static**" and "**final**" keywords define a constant

```
public class EmployeeRecordDefaults
{
    static final int    NUMBER = 0;
    static final char   TYPE   = 'R';
    static final String NAME   = "Joe Public";
    static final String ADDRESS = "1 Young St";
    static final Date   HIRED  = new Date();
    static final BigDecimal SALARY
        = new BigDecimal("30000.00");
}
```

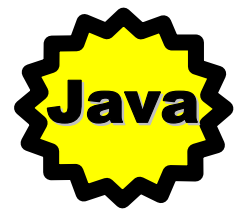
Java

```
D*..1....+....2....+....3....+....4....+....5
D*EmpRcdDFT       DS
D numberDFT       C                   CONST(0)
D typeDFT         C                   CONST('R')
D nameDFT         C                   CONST('Joe Public')
D addressDFT      C                   CONST('1 Young St')
D hiredDFT        C                   CONST(D'1999-12-31')
D salaryDFT       C                   CONST(30000)
```

RPG

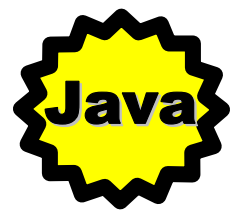
More on boolean

- Can be assigned true or false (reserved words keywords) in Java
  - Boolean myFlag = true;
- Can be assigned an expression:
  - Boolean myFlag = (rate > 10);
- Can be used in place of an expression:
  - if (rate > 10) ... *** or ***
  - if (myFlag)
- Can be negated:
  - myFlag = !myFlag;
  - while (!myFlag) ...



What about packed?

- No packed decimal data type in Java
- Could use float or double, but precision is a problem for "fixed decimal" numbers
- Answer: `BigDecimal` class
  - A class, not a built-in "primitive" data type
  - Software simulation of fixed decimal numbers
  - Unlimited "precision" (total number of digits)
  - Program control over "scale" (number of decimal digits)
  - Full complement of math operators, via methods
- Bottom line: it does the job!
- See also: `BigInteger` class



BigDecimal class (1 of 3)

Using float

Java

```
public class TestFloat
{
    public static void main(String args[])
    {
        double unitCost = 1234.56;
        double discount = 0.23;
        System.out.println("unitCost = " + unitCost);
        unitCost = unitCost - unitCost * discount;
        System.out.println("saleCost = " + unitCost);
    }
}
```

```
unitCost = 1234.56
saleCost = 950.6111999999999
```

```
unitCost = 1234.56
unitCost after discount = 950.6112
unitCost after setScale = 950.61
```

Using BigDecimal

```
public class TestBD
{
    public static void main(String args[])
    {
        BigDecimal unitCost = new BigDecimal("1234.56");
        BigDecimal discount = new BigDecimal(".23");
        System.out.println("unitCost = " + unitCost);
        unitCost = unitCost.subtract( unitCost.multiply(discount) );
        System.out.println("unitCost after discount = " + unitCost);
        unitCost = unitCost.setScale(2, BigDecimal.ROUND_HALF_UP);
        System.out.println("unitCost after setScale = " + unitCost);
    }
}
```

BigDecimal class (2 of 3)

`java.math.BigDecimal`

Methods	Description
BigDecimal(double) BigDecimal(String)	Constructors. Can create from a double primitive value or a String literal, such as `12345.67`.
BigDecimal abs(), negate()	Return absolute or negative version of this BigDecimal.
BigDecimal add(BigDecimal) BigDecimal subtract(BigDecimal)	Add or subtract given BigDecimal to or from this BigDecimal, returning result. Scale = max of two scales.
BigDecimal divide(BigDecimal, int)	Divide this BigDecimal by given BigDecimal, specifying rounding constant, such as ROUND_HALF_UP. Scale is the same as before the operation.
BigDecimal multiply(BigDecimal)	Multiply this BigDecimal by given BigDecimal returning resulting BigDecimal. Scale = sum of the two scales.
BigDecimal setScale(int,int)	Force to given scale (# of decimals). Second parm is rounding constant, such as ROUND_HALF_DOWN.
BigDecimal movePointLeft(int) BigDecimal movePointRight(int)	Move decimal point left or right by given amount.
int compareTo(BigDecimal)	Compare this BigDecimal to given one, returning -1, 0, or 1 indicating less than, equal, or greater than.
boolean equals(BigDecimal)	Compare for equaling, returning true or false.
longValue(), intValue(), doubleValue(), floatValue(), toString()	Convert to various primitive types or to a printable String.

BigDecimal class (3 of 3)

- Note that BigDecimal is an *immutable* class:
  - Specially designed so that no method changes the stored values.
  - Rather, every method returns a totally new BigDecimal object.
  - Be careful! Remember to always assign the result back.
  - One other important immutable class in Java: String.
    - Covered in Chapter 7

Modifier keywords

```
private int yourVariable = 980;
```

Modifiers	Description
public, private, or protected	Accessibility of the variable. Public means all can access, private means only this class, and protected means this class or those which extend it. The default is package, meaning this class or others in this package can access it.
static	This variable is initialized only once and has only one value, regardless of the number of instances of this class.Can be accessed via classname.variable versus object.variable. Only class level variables support this. Static variables are known as class variables.
static final	This variable is a constant and cannot be changed. By some conventions, constants always have uppercase names to distinguish them from non-constants.

RPG field scoping

- RPGIII
  - Every field is global to the program.
- RPG IV (V3R2, V3R6, and above)
  - Fields defined in mainline area of the RPG main C-Spec are accessible by all procedures as global fields to the program.
  - Fields declared inside RPG procedures are only accessible by that procedure.
- You can also use the new EXPORT keyword in RPG IV.
  - Allows other modules in this *PGM or *SRVPGM to read or update this field

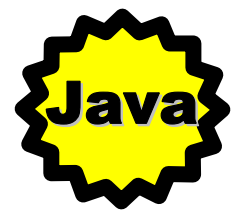
Java field scoping (1 of 2)

- Variables can be declared at the class or method level.
  - Parameter variables are considered local.
  - Global variables in RPG IV == class level variables in Java.
  - Local fields in RPG IV == local variables in Java.
- In addition, Java has blocks, as in:


```
int myVariable;
myVariable = 10;
{
  int mySecondVariable;
  mySecondVariable = 20;
}
System.out.println("mySecondVariable =  " + mySecondVariable);
```

Java field scoping (2 of 2)

- Most commonly used in an `if` statement or `in` looping blocks:
  - if - else blocks
  - for loops
  - while loops
  - do-while loops




```
for (int index = 0; index < 10; index++)
{
   System.out.println("index = " + index);
}
```

Example

```
public class Widget
{
   public  static final int TYPE1 = 1; // constant
   public  static final int TYPE2 = 2; // constant
   public  static       int nextID= 0; // class
   private              int id;        // instance
   private              int type  = 0; // instance
   public Widget() // Constructor. Note name == class name
   {
       id = nextID; // references instance and class vars
       nextID = nextID + 1;
   }
   public boolean setType(int newType)
   {
       boolean inputOK = true; // local
       if ((newType >= TYPE1) && (newType <= TYPE2))
         type = newType; // references instance variable
       else
         inputOK = false;
       return inputOK;
   }
   public String toString()
   {
       String retString; // local
       retString = "Type = " + type + ", ID = " + id;
       return retString;
   }
} // end Widget class
```

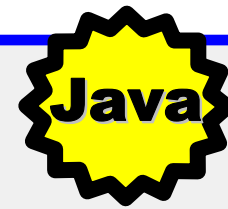
Class variables

Instance variables

Parameter variable

Local variable

Testing the example




```
public class TestWidget
{
    public static void main(String args[])
    {
        Widget.nextID = 1000; // set class variable
        Widget myWidget = new Widget(); // object 1
        myWidget.setType(Widget.TYPE1); // call method
        Widget myWidget2 = new Widget();// object 2
        myWidget2.setType(Widget.TYPE2);// call method

        System.out.println(myWidget);  // calls toString method
        System.out.println(myWidget2); // calls toString method
    }
} // end TestWidget class
```

Result

```
C:\JAVA>JAVA TestWidget
 Type = 1, ID = 1000
 Type = 2, ID = 1001
```

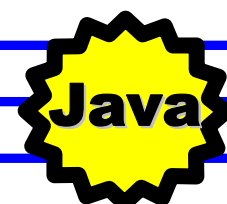
Literals (1 of 2)

RPG Type	Example	Java Type	Example
character 1	`'a' or X'7D'`	**char**	'a'
character n	`'abc'`	**String**	"ABC" or "c:\\mydir"
graphic	`G'oK1K2i'`		
indicator	`'0' or '1' or *OFF or *ON`	**boolean**	**true** or **false**
date	`D'97-12-11'`		See Date and Time chapter
time	`T'11:33;01'`		See Date and Time chapter
timestamp	`Z'1997-12-11.33.01'`		See Date and Time chapter
basing pointer	`*NULL or %ADDR(myVar)`	**object reference**	**null** or **new** MyClass()
procedure pointer	`*NULL or %PADDR(myProc)`		

Literals (2 of 2)

- Java allows special characters:

Escape Sequence	Description
\n	newline character
\t	tab
\b	backspace
\r	carriage return
\f	form feed
****	backslash
\'	single quote
\"	double quote
\ddd	octal number, not to exceed 377
\uxxxx	Unicode number. Must be 4 digits



Numeric literals

RPG Type	Example	Java Type	Example
		byte	10 or -10
integer 5	10 or -10	**short**	10 or -10
integer 10	10 or -10	**int**	10 or -10
		long	10 or 10**L** or -10 or -10**L**
unsigned	10 or 20	**n/a**	
binary	10 or 10.1	**n/a**	
packed	10 or 10.1	**n/a**	
zoned	10 or 10.1	**n/a**	
float 4	10 or .12 or 1234.9E12	**float**	10.0**f** or 12.1**f** or 1.234**E**12**f**
float 8	10 or 12 or 1234.9E12	**double**	10.0 or 10.0**d** or 12.1 or 1.234**E**12

Note: All decimal point literals considered double unless they end with letter **f.**

Casting in RPG IV

```
...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+...
*************** Beginning of data ************************
     FQSYSPRT   O    F   80        PRINTER OFLIND(*INOV)
     D DS1             DS
     D  int5                          5I 0 INZ(25)
     D  BIN9                          9B 0 INZ(22)
     D  ZONE9                         9S 0 INZ(30)
     D  PACK9                         9P 0 INZ(40)
     D*-------------------------------------------------
     C                   MOVE      BIN9          INT5
     C                   EXCEPT    RESULT
     C                   MOVE      PACK9         INT5
     C                   EXCEPT    RESULT
     C                   MOVE      ZONE9         INT5
     C                   EXCEPT    RESULT
     C                   MOVE      *ON           *INLR
     OQSYSPRT   E            RESULT
     O                       INT5                15
 ***************** End of data **************************
```

- Casting is implicit in RPG; no explicit syntax required.

Casting in Java

- RPG: Casting is always implicit.
- Java: Casting is implicit only if target type has larger precision than source type; otherwise, explicit casting is required.
- Explicit cast syntax: (target-type)source-value.


```
public class TestCast
{
    public static void main(String args[])
    {
        short sValue = 10;
        long  lValue = 30;
        lValue = sValue; // implicit
        sValue = (short)lValue;// explicit
    }
}
```

What about overflow?

- What happens if source will not fit in target?
  - Nothing; no overflow indicators are in Java.
  - Casting syntax presumes you know what you are doing.
- Your job is to check for overflow first before casting:
  - Use MIN_VALUE and MAX_VALUE constants in Long, Short, Integer, Byte "wrapper" classes.


```
if ((lValue <= Short.MAX_VALUE) &&
    (lValue >= Short.MIN_VALUE))
  sValue = (short)lValue; // cast
else
  // overflow/underflow error...
```

Casting in expressions

- You can cast an expression, too, in order to force result to a particular type.
  - Else, resulting type is largest of all the operands
    - Remember float literals are double precision unless they end in *f or F*, integer literals are int types unless end in *l or L.*


```
public static void main (String args[])
{
    float  currPrice   = 12.5f; // Force literal to float
    double perCentSale = 22.0;
    float  finalPrice  =  // have to cast down to float
      (float)(currPrice - (currPrice * (perCentSale / 100)));
}
```

Force a double expression to a single (float) value

Casting summary table

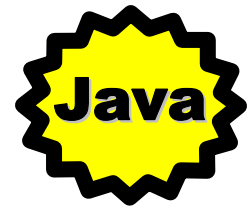
	byte	char	short	int	long	float	double
byte	No	Cast[1]	No	No	No	No	No
char	Cast	No	Cast[1]	No	No	No	No
short	Cast	Cast	No	No	No	No	No
int	Cast	Cast	Cast	No	No	No	No
long	Cast	Cast	Cast	Cast	No	No	No
float	Cast	Cast	Cast	Cast	Cast	No	No
double	Cast	Cast	Cast	Cast	Cast	Cast	No

read left to right

- Cast[1] -> possible loss of sign:
  - Signed to unsigned or vice versa
  - Char is 2-byte unsigned data type; all others are signed

Class wrappers

- Primitive types have wrappers
  - Classes in java.lang package
    - Objects hold primitive value within them.
    - Also methods to convert from and to primitives.
  - Sometimes you will need them
    - Such as for vectors as you will see in Chapter 6.
    - They also have handy methods and constants.



Primitive	Wrapper
byte	Byte
short	Short
int	Integer
long	Long

Primitive	Wrapper
char	Character
boolean	Boolean
float	Float
double	Double

Integer class

`java.lang.Integer`

Methods	Description
Integer(int), Integer(String)	Constructors. Create from an int or String literal such as
int compareTo(BigDecimal)	Compare this BigDecimal to given one, returning -1, 0 or 1 indicating less than, equal to, or greater than
boolean equals(Object)	Compare for equaling, returning true or false
int hashcode()	Returns a unique hash value for this object
byteValue(), shortValue(), longValue(), intValue(), doubleValue(), floatValue(), toString()	Convert to various primitive types or to a printable String

Static Methods	Description
Integer decode(String), Integer valueOf(String)	Parses decimal, integer, or hexadecimal literal into an Integer object
int parseInt(String)	Parses a string such as "-1234" and returns it as a int
String toBinaryString(int), toOctalString(int), toHexString(int), toString(int)	Converts int primitive value to a printable string, displaying value in binary, octal, hexadecimal, or decimal format

Public Constants	Description
int MAX_VALUE, int MIN_VALUE	Largest int value and smallest int value (2147483647, -2147483648)

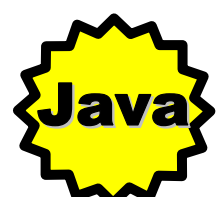
Class wrapper methods

Class	From a String	To a String	To a Primitive
Boolean	valueOf(String)	toString()	booleanValue()
Byte	valueOf(String) decode(String)	toString()	byteValue(), doubleValue(), floatValue(), intValue(), shortValue(), longValue()
Character		toString()	charValue(), getNumericValue()
Double	valueOf(String)	toString()	doubleValue(), byteValue(), floatValue(), intValue(), longValue(), shortValue()
Float	valueOf(String)	toString()	floatValue(), byteValue(), doubleValue(), intValue(), longValue(), shortValue
Integer	valueOf(String) parseInt(String)	toString() toBinaryString(long) toHexString(long) toOctalString(long)	intValue(), byteValue(), doubleValue(), floatValue(), longValue(), shortValue()
Long	valueOf(String) parseLong(String)	toString() toBinaryString(long) toHexString(long) toOctalString(long)	longValue(), byteValue(), doubleValue(), floatValue(), intValue(), shortValue()
Short	valueOf(String) parseShort(String)	toString()	shortValue(), byteValue(), doubleValue(), floatValue(), intValue(), longValue()

Example: Convert to float

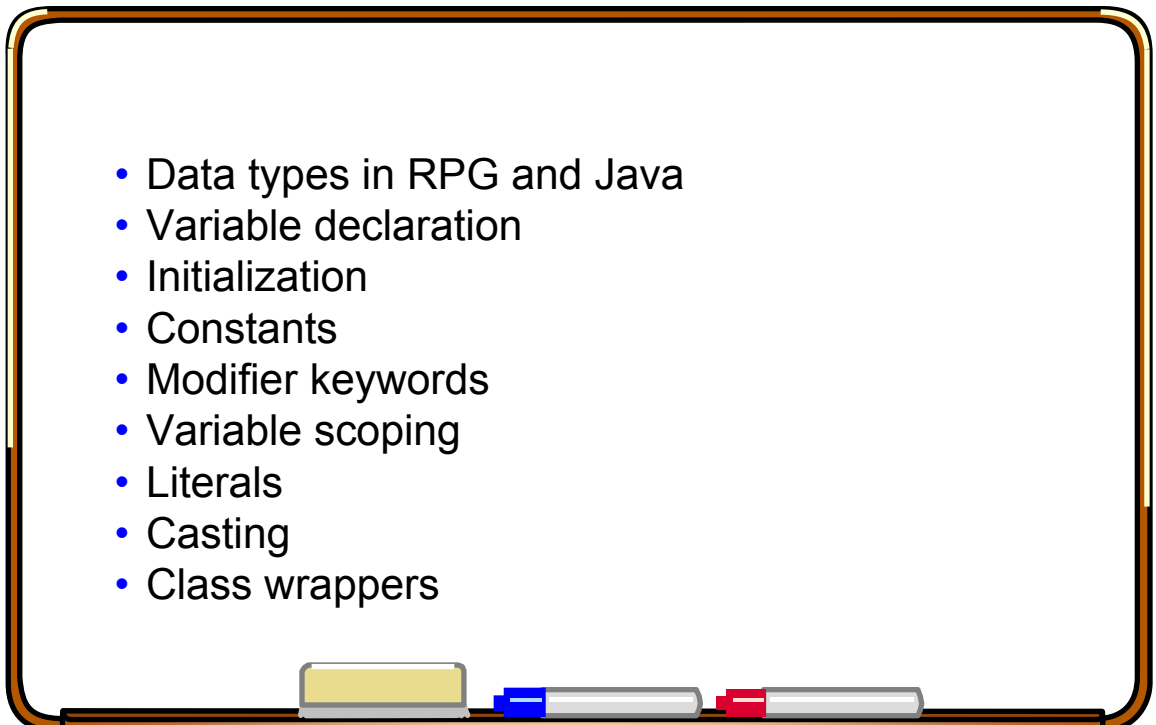
```
public class TestConvertFloat
{
   public static void main(String[] args)
   {
     Float floatObject;

     if (args.length != 1)
       return;

     try
     {
      floatObject = Float.valueOf(args[0]);
     }
     catch(NumberFormatException exc)
     {
      System.out.println("Invalid input!");
      return;
     }

     float floatValue = floatObject.floatValue();

     System.out.println("input = " + floatValue);
   }
}
```

```
C:\JAVA>javac TestConvertFloat.java
C:\JAVA>java  TestConvertFloat 1.2
input = 1.2
C:\JAVA>java  TestConvertFloat 1.02
input = 1.02
C:\JAVA>java  TestConvertFloat 0000.12
input = 0.12
C:\JAVA>java  TestConvertFloat 1.2e4
input = 12000.0
C:\JAVA>java  TestConvertFloat -1.2e4
input = -12000.0
C:\JAVA>java  TestConvertFloat abcdef
Invalid input!
```



we cover try - catch in Chapter 10

Topics covered



Unit summary

Having completed this unit, you should be able to:

- Describe how Java declares and represents program data
- Use Java syntax to declare and initialize program variables to store data
- Distinguish variables of different access scope
- Use various classes in the JDK to manipulate data