Random Class

An instance of the Random can be used to generate a stream of random values

<u>Typical process:</u>

1. Create a Random object
2. Use the object to get random values. Use one of:

nextInt()	returns a random integer
nextInt(max)	returns an integer random value in **[**0, … max **)**
nextDouble()	returns a random value in **[**0.0, 1.0 **)**
nextBoolean()	returns a random value from {true, false}

Note: need an import statement

```
import java.util.Random;
```

Random Class

Examples

1. Simulate a 3-way choice: rock/paper/scissors – use `nextInt(3)`
2. Simulate coin toss – use `nextInt(2)`
3. Simulate coin toss – use `nextBoolean()`
4. Simulate tossing a coin 100 times counting number of heads

Random class – Rock/paper/scissors

Example 1. code to display a random Rock-Paper-Scissors:

```
// three outcomes, all equally likely
Random rand = new Random();
// nextInt(3) produces a value from {0, 1, 2}
switch( rand.nextInt(3) ) {
case 0:
        System.out.println("Rock");
        break;
case 1:
        System.out.println("Paper");
        break;
case 2:
        System.out.println("Scissors");
        break;
}
```

Random class – toss a coin

Example 2. code to display a random coin toss:

```
// two-sided coin: heads/tails
Random rand = new Random();
// nextInt(2) produces values from {0, 1}
switch( rand.nextInt(2) ) {
case 0:
        System.out.println("Heads");
        break;
case 1:
        System.out.println("Tails");
        break;}
```

Tossing coins using Booleans → next slide

Random class – toss a coin using booleans

Example 3. code to display a random coin toss:

```
// two-sided coin: heads/tails
Random rand = new Random();
// nextBoolean() produces values from {true, false}
// cannot switch on booleans → use if-else
if( rand.nextBoolean() )
   System.out.println("Heads");
else
   System.out.println("Tails");
```

Random class

Example 4: Simulate tossing a coin 100 times (page 117 in text)

```
public class TossCoin
{
public static void main ( String [] args )
{
    int heads = 0; // counter for heads
    System.out.print("\ n100 tosses : ");
    Random g = new Random () ;

    for (int i=0; i<100; i++)
        if( g.nextBoolean() ) heads ++;

    System.out.println("\ nHeads : "+ heads
           +"\ nTails : "+(100 - heads ) ) ;
    }
}
```

Random Number Generators

ASIDE: What does a random number generator look like?

https://en.wikipedia.org/wiki/Linear_congruential_generator

```
nextValue ← (a * previousValue + c) mod m
```

Java: next ← (25214903917*previous + 11) mod 2^{31}
uses 48-bit values at each iteration but returns
the 32 most significant bits

Character class

An instance of the Character class is not required

Character contains many useful utility methods

Method	Description
getNumericValue(...)	Returns the int value that the specified character represents.
isDigit(...)	Determines if the specified character is a digit.
isLetter(...)	Determines if the specified character is a letter.
isWhitespace(...)	Determines if the specified character is white space
toLowerCase(...)	Converts the character argument to lowercase
toUpperCase(...)	Converts the character argument to uppercase

Character class

Examples

Notice in the code how the Character class methods are specified when there is no object … this may seem odd, but …

1. Detecting letters, digits

2. Getting a numeric value of a character that is a digit

3. Checking a control number for validity
    - Suppose all characters must be numeric
    - Consider exercise 6 on page 125

Character class

Example 1

```
public class CharacterTypes
```

A line of text is examined, character-by-character, to determine the character's type where type is one of {letter, digit, other}

Character methods used:

`isLetter(…)` returns `true` if the character is a letter

`isDigit(…)` returns `true` if the character is a digit

No instance of Character is used which means the methods are called using statements of the form

```
If ( Character.isDigit(c) ) System.out.println(…
```

Prefix *Character.* Is needed to reference a static method of the Character class

The method to execute is *isDigit*

The argument passed to *isDigit* is the character *c*

Character class

Example 1

```
public class CharacterTypes{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter a line: ");
        String line = kb.nextLine();
        // characters are examined one-by-one
        for (int i = 0; i < line.length(); i++){
            char c = line.charAt(i);
            if( Character.isLetter(c) )
               System.out.println(i+"\t"+c+"\t\tletter");
            else if( Character.isDigit(c) )
               System.out.println(i+"\t"+c+"\t\tdigit");
            else
               System.out.println(i+"\t"+c+"\t\tother");
        }
    }
}
```

Character class

Example 2

```
public class SumNumericValues
```

A line of text is examined, character-by-character, and the sum of the numeric characters is calculated

Character methods used:

`getNumericValue(…)` returns the `int` value the character represents

`isDigit(…)` returns `true` if the character is a digit

No instance of Character is used which means the methods are called using statements of the form

```
sum += Character.getNumericValue(c) ;
```

The *Character* class

The method to execute is *getNumericValue*

The argument passed to *getNumericValue* is *c*

Character class

Example 2

```
public class SumNumericValues
{
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("\nEnter a line: ");
        String line = kb.nextLine();
        int sum = 0;
        // characters are examined one-by-one
        for (int i = 0; i < line.length(); i++){
            char c = line.charAt(i);
            if( Character.isDigit(c) ){
                sum += Character.getNumericValue(c);
            }
        }
        System.out.println("sum = \t"+sum);
    }
}
```

Character class

Example 3

```
public class ValidateStudentNumber
```

A student number, stored as a character string, is examined character-by-character to determine if all characters are digits.

Character methods used:

`isDigit(…)` returns `true` if the character is a digit

No instance of Character is used which means the methods are called using statements of the form

```
If ( ! Character.isDigit(c) ) valid = false ;
```

The *Character* class

The method to execute is *isDigit*

The argument passed to *isDigit* is *c*

Character class

Example 3

```
public class ValidateStudentNumber
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        System.out.println("Enter a number: ");
        String number = kb.next();
        // characters are examined one-by-one
        boolean valid = true;
        for (int i = 0; i < number.length(); i++){
            char c = number.charAt(i);
            if(! Character.isDigit(c) ) valid = false;
        }
        if (valid) System.out.println("Valid");
        else System.out.println("Invalid");
    }
}
```

Scanner class

The Scanner class is useful when you need to parse some *stream* that is said to contain *tokens*.

For ACS-1903 we are concerned with tokens that comprise a sequence of characters delimited by whitespace (space, tab, line feed characters)

We can easily create a Scanner object that is associated with one of:

- System.in
- a string
- a file

} Streams containing tokens

Examples

```
Scanner s = new Scanner(System.in);
Scanner s = new Scanner(s);   //s is of type String
Scanner s = new Scanner(f);   //f is of type File
```

Scanner class

Scanner methods

next()	get next token
nextBoolean()	get next …
nextInt()	get next …
nextDouble()	get next …
nextLine()	get next …
hasNext()	returns true if there is a token available
hasNextLine()	returns true if …
hasNextBoolean()	returns true if …
hasNextInt()	returns true if …
hasNextDouble()	returns true if …

Scanner class

Examples

1. Read a file named Readme.txt … this file is in every BlueJ project.
   Three import statements

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException
```

   Exceptions are a topic in ACS-1904.
   There is a `throws` clause – something can go wrong when reading and writing files – a topic in ACS-1904

2. Scanning a string for tokens.
   One import statement

```
import java.util.Scanner;
```

Scanner class

Example 1

```
public class DisplayReadme
```

The file Readme.txt is read, line-by-line, until there are no lines left.

Scanner methods used:

`f.hasNext()` returns `true` if there is another token in the stream `f`

`f.nextLine()` returns the next line in `f`

A Scanner object is needed to provide a reference to the file and the current location in the file. Consider the statement:

```
Scanner f = new Scanner( new File("Readme.txt"));
```

The *Scanner* object is *f*

Create a *Scanner* object for a *file*

The file *Readme.txt*

Scanner class

Example 1

```
public class DisplayReadme
{
    public static void main(String[] args)
    throws FileNotFoundException
    {
        Scanner f = new Scanner(new File("Readme.txt"));
        int i=1;
        System.out.println("<<<< File Readme.txt >>>>");
        while ( f.hasNext() ){
            String line = f.nextLine();
            System.out.println((i++)+" "+line);
        }
        System.out.println("<<<< end of listing >>>>");
    }
}
```

Scanner class

Example 2

```
public class ScanString
```

A string with whitespace separating the tokens is parsed
– each token is displayed on a separate line.

Scanner methods used:

`s.hasNext()` returns `true` if there is another token in the stream `s`

`s.next()` returns the next token in `s`

A Scanner object is needed that provides a reference to the file and the current location in the file. Consider the statement:

```
String token = s.next() ;
```

The next token is assigned to a variable of type String

The method *next()*

The *Scanner* object named *s*

Scanner class

Example 2

```
public class ScanString
{
    public static void main(String[] args)
    {
        String sample = "one    two \tthree";
        Scanner s = new Scanner(sample);
        System.out.println("<<<<"+sample+">>>>");
        while (s.hasNext()){
            String token = s.next();
            System.out.println(token);
        }
        System.out.println("<<<< end of tokens >>>>");
    }
}
```

Math class

`Math` class is a utility class

- You cannot create an instance of `Math`
- All references to constants and methods will use the prefix `Math.`

- Contains constants, π and e

  Names of these are `PI` and `E`

  Java convention is to name constants using capital letters.

Math class

Methods

`pow(…)`	Raise the first argument to the power specified in second argument e.g. `Math.pow(x,3)`
`abs(…)`	Returns absolute value of its argument
`max(…)`	Returns the larger of two `int` or `double` arguments
`min(…)`	Returns smaller …
… many more	

Math class

Example

1. Find the largest of 3 int values
   As there is no object note the use of the prefix ***Math.***

Math class

Example 1

```
public class FindMax
```

The larger of 3 integers is determined. The `max` method is used twice.

Scanner methods used:

`Math.max()` returns the larger of two values passed in as arguments

`Math` is a utility class with static methods.

Consider the statement:

```
Math.max(j,k)
```

Two arguments, `j` and `k`, passed in to `max`

The method `max`

The `Math` class

Math class

Example 1

```
public class FindMax
{
    public static void main(String[] args){
        Scanner kb = new Scanner(System.in);
        System.out.println(
            "Please enter 3 int values");
        int i = kb.nextInt();
        int j = kb.nextInt();
        int k = kb.nextInt();
        int mx = Math.max(i, Math.max(j,k));
        System.out.println("largest is "+mx);
    }
}
```

Note how Math.max(…) is used twice

Integer class

`Integer` class is a utility class

- Many methods are static

  you do not need an object of type Integer.

  The prefix `Integer.` is used for these.

- Contains constants, `MAX_VALUE` and `MIN_VALUE`

  Again … Java convention is to name constants using capital letters.

Integer class

Methods

`max(…)`	Returns the larger of two `int` arguments
`min(…)`	Returns smaller …
`parseInt(…)`	Parses the string argument expecting that argument to be a valid decimal integer. Correction to notes: E.g. `parseInt(" 23 ")` *no spaces* Should be: E.g. `parseInt("23")`
…	

to ensure there are no leading or trailing spaces in a string one can use the `trim()` method

```
String xx = ...
xx = xx.trim();
```

Integer class

Example

Read lines of text from System.in
Each line is *parsed* according to the expected format:
<name of an item><comma><quantity as integer>

Examples of such lines:
monitor,45
laptop,55

Integer class

Example

```
public class TotalQuantity
```

Integer methods used:

`parseInt(…)` returns the integer represented by a character string

`Integer` is a utility class with static methods.

Consider the statement:

```
int qty = Integer.parseInt(qtyAsString);
```

Int variable

The `Integer` class

The method `parseInt`

String containing an integer

Integer class

Example 1

If `qtyAsString` does not have a valid integer value the program will terminate with an error

```
public class TotalQuantity
{
    public static void main(String[] args)
    {
        Scanner kb = new Scanner(System.in);
        int totalQty = 0;
        for (int i = 0; i < 4; i++){
            System.out.print("Enter next line: ");
            String line = kb.nextLine();
            int commaAt = line.indexOf(",");
            String qtyAsString = ine.substring(commaAt+1);
            int qty = Integer.parseInt(qtyAsString);
            totalQty += qty;
        }
        System.out.println("total = "+totalQty);
    }
}
```

Wrapper classes

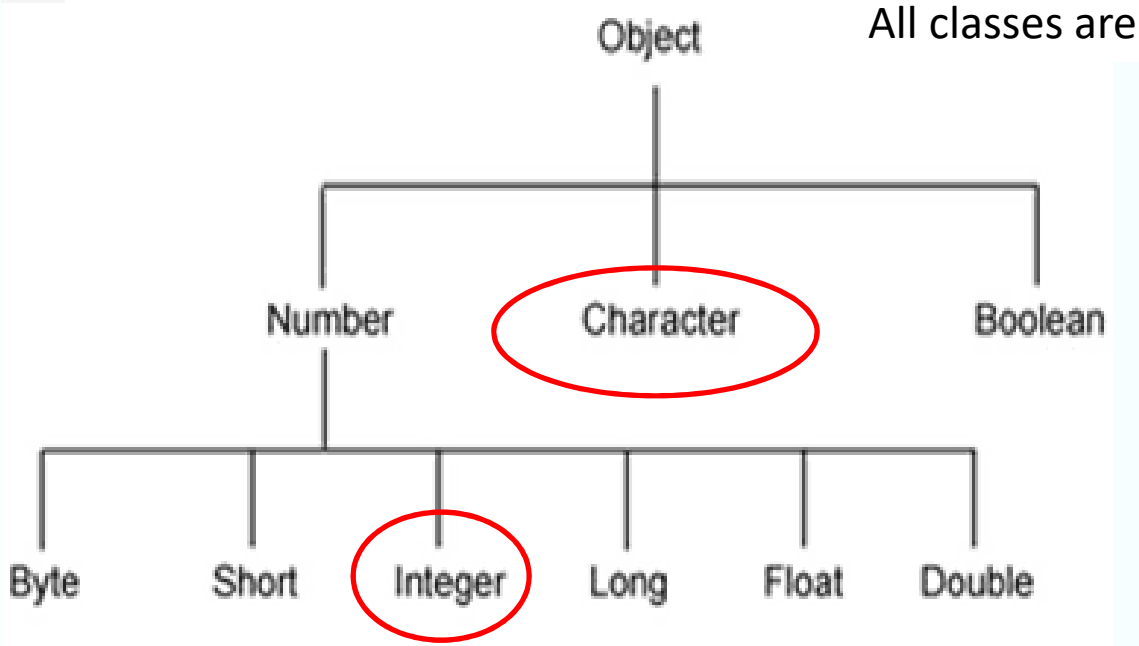
With similarity to the `Integer` class, there are classes for other types … these types of classes are called *wrapper* classes.

These are called wrapper classes because you instantiate an object and *wrap* a primitive value inside

```
Double
Boolean
Byte
Character
Float
Long
Short
```

Aside: Hierarchy of classes

A topic in ACS-1904 is class hierarchies. For example:



All classes are subclasses of Object