Chapter 3: Using Classes and Objects

CS 121

Department of Computer Science
College of Engineering
Boise State University

November 2, 2015

Chapter 3 Topics

- Part 0: Intro to Object-Oriented Programming

  Go to part 0
- Part 1: Creating Java objects

  Go to part 1
- Part 2: The `String` class

  Go to part 2
- Part 3: The `Random` class

  Go to part 3
- Part 4: The `Math` class

  Go to part 4
- Part 5: Formatting output

  Go to part 5
- Part 6: Wrapper classes and autoboxing

  Go to part 7

Brief Intro to Object-Oriented Programming

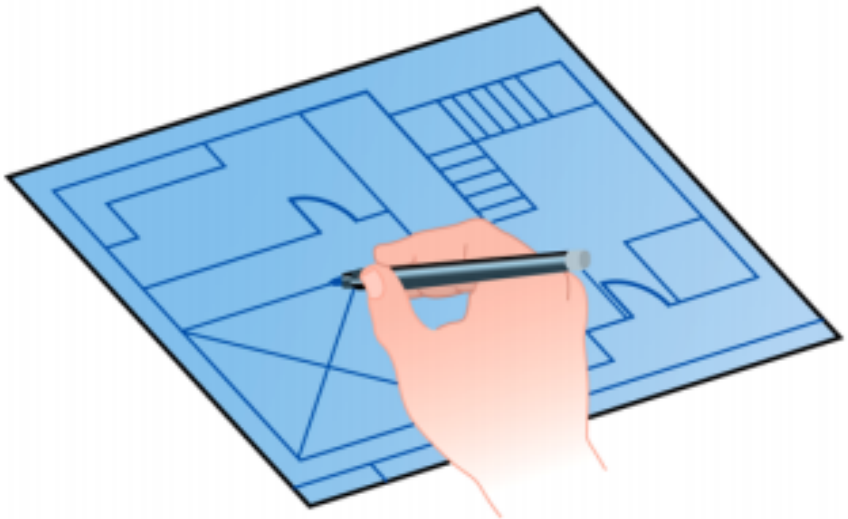
- Java is an object-oriented programming language.
- Objects are used to represent real-world things.
- Objects have state and behaviors.
  - Dog 
    - state: name, breed, color, age, hungry, etc.
    - behavior: walk, run, bark, lick, fetch
  - String `"Hello World!"`
    - state: length, characters
    - behavior: get length, equals, sub-string, compare to, to upper case, etc.

Classes

- Objects are defined by classes.
- Multiple objects can be created from the same class.
- Define variables to represent state.
- Define methods to define behaviors.

Classes and Objects

- We can think of a class as the blueprint of an object.
- One blueprint to create several similar, but different, houses.



Classes and Objects

- An object is an instance of a class.
- Objects are encapsulated to protect the data it manages.
- Classes can be created based on another class using inheritance (You will see more of inheritance in CS 221).

Creating Objects

- **Recall:** A variable holds either a primitive type or a reference to an object.
- A variable referring to an object is known as a reference variable.
- The class name of the object is used as the type in the declaration statement.

```
String courseName;
Scanner keyboardInput;
```

- This declaration *does not create* an instance of the object.
- It is just a reference to an actual object stored in memory.
- The object must be *explicitly* created.

Instantiating an Object

- We use the `new` operator to create a new object.

```
String courseName = new String("CS 121");
```

- This calls the `String` constructor – a special method that sets up the object.
- The new object is an instance of the class.

Instantiating String Objects - A special case

- A quick note.... We don't have to use the `new` operator to create a `String`.
- We can use string literals.

```
String courseName = "CS 121";
```

- This is *only* supported for `String` objects (because they are so frequently used). The Java compiler creates the object for us as a convenience.

Invoking Methods of an Object

- After we instantiate an object, we can use the dot operator to invoke its methods.

```
String courseName = new String("CS 121");
int length = courseName.length();
```

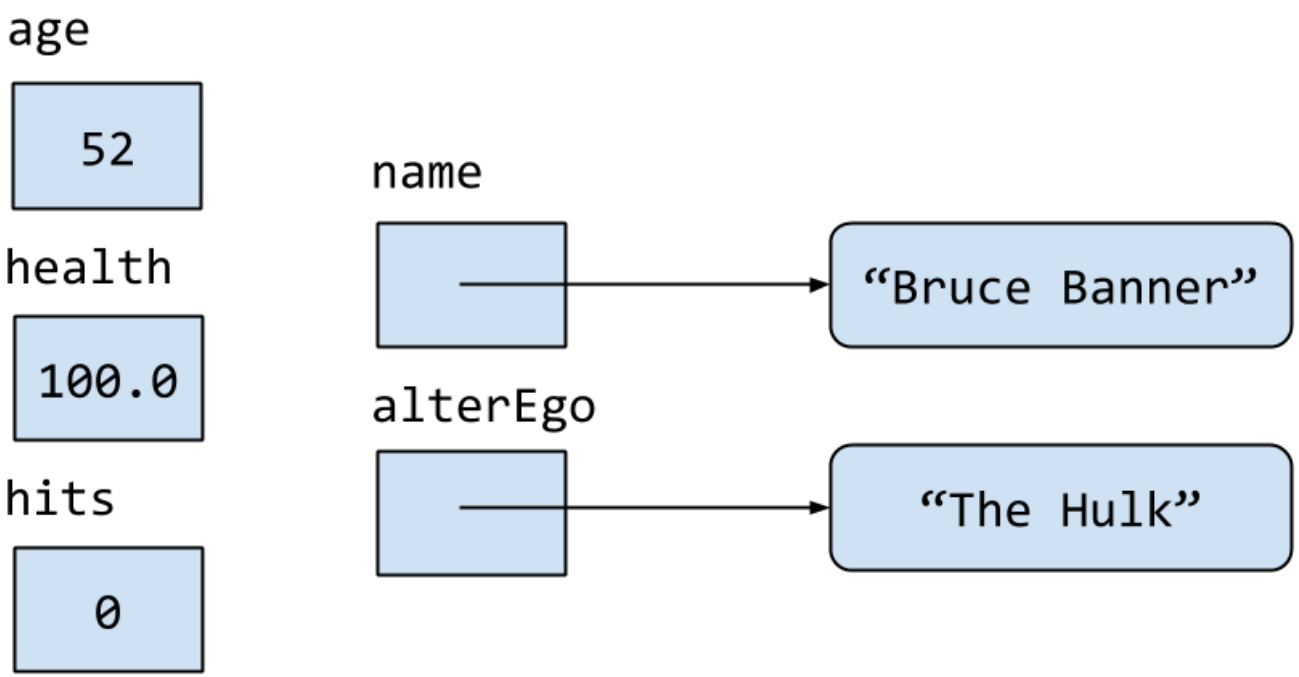
- Methods may return values that can be used in an assignment or expression.
- Invoking an object's method can be thought of as asking the object to do something.

Object References

- Primitive variables and object variables store different information.
- Primitive variables (e.g. `int`, `char`, `boolean`) contain the *value* itself.
- Object variables (e.g. `String`) contain the *address* of the object it references.

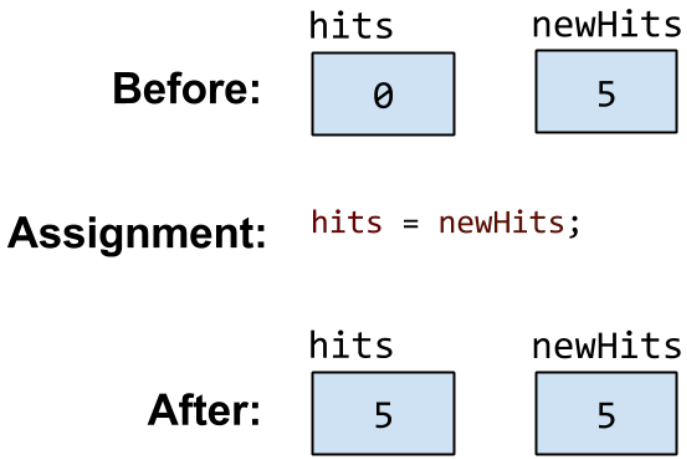
Object References: The Hulk

```
int age = 52;
String name = new String("Bruce Banner");
String alterEgo = "The Hulk";
double health = 100.0;
int hits = 0;
```



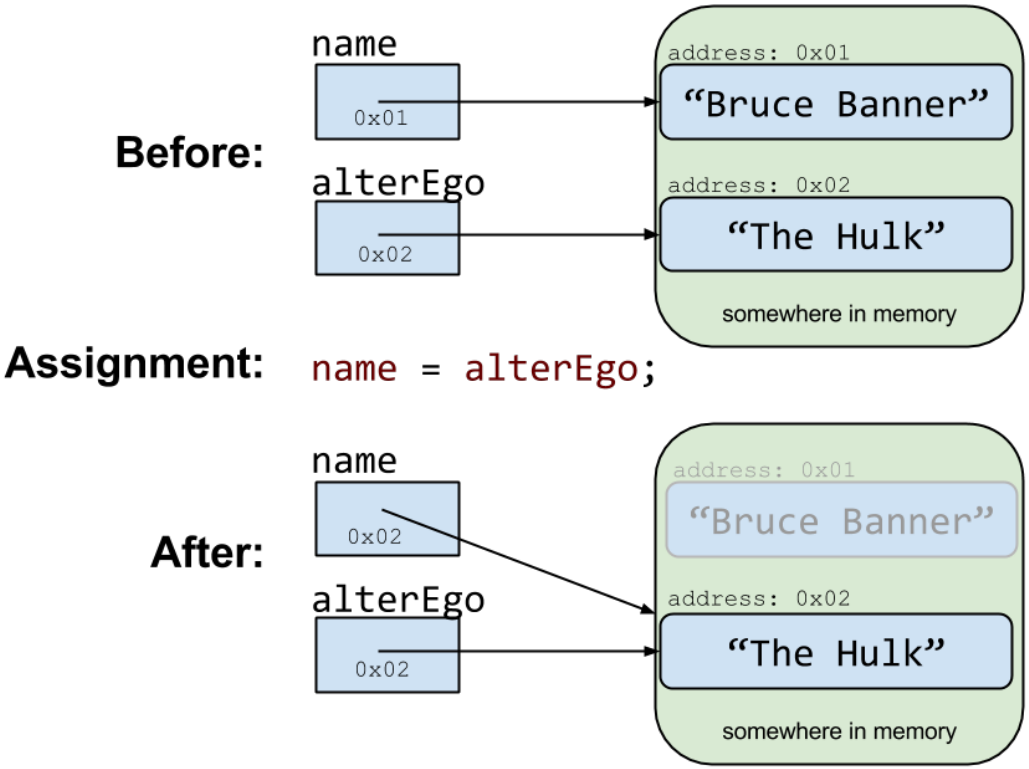
Assignment Revisited

- **Recall:** The act of assignment takes a copy of a value (the *Right-Hand-Side) and stores it in the target variable (the* Left-Hand-Side).
- For primitive types, the *value* of the variable is copied.



Assignment Revisited

- For objects, the *address* of the object is copied.



Aliases

- Two or more references that refer to the same object are aliases of each other.
- A single object can be accessed using multiple references.
- This is useful, but can cause issues if not managed properly.
- *Changing an object through one reference changes it for all of its aliases, because there is really only one object stored in memory.*

Example: Swapping Two Variables

- Suppose we have two `int` variables that we want to swap. We need a temporary variable to complete the swap:

```
int n1 = 100, n2 = 200;
```

```
int tmp = n1; n1 = n2; n2 = temp;
```

- Suppose we have two `String` variables that we want to swap. We need a temporary `String` variable to complete the swap. This is a good example of using aliases.

```
String s1 = "hello";
String s2 = "goodbye";
String tmp = s1; s1 = s2; s2 = tmp;
```

Garbage Collection

- If there are no variables that refer to an object, the object is inaccessible and referred to as garbage.
- Java performs automatic garbage collection in the background, reclaiming the memory used by garbage objects for future use.
- In some languages, the programmer is responsible for freeing the memory used by garbage objects.

The Java API

- The Java API is the standard class library that provides a large collection of pre-built classes that we can use in our programs.
- API = **A**pplication **P**rogramming **I**nterface
- Before writing our own classes, we will practice using several classes that are part of the Java API.
- The classes of the Java API are organized into packages. Java comes with hundreds of packages and tens of thousands more can be obtained from third-party vendors.
- Java API docs:
  `http://docs.oracle.com/javase/8/docs/api/`

Selected Java Packages

Package	Provides
`java.lang`	Fundamental classes
`java.util`	Various useful utility classes
`java.io`	Classes for variety of input/output functions
`java.awt`	Classes for creating graphical user interfaces and graphics
`java.swing`	Lightweight user interfaces that extend AWT capabilities
`java.net`	Networking operations
`java.security`	Encryption and decryption

Import Declarations

- When you want to use a class from a Java API package, you need to import the package.

```
import java.awt.Graphics;
```

- To import *all* classes in a package, you can use the wild card character (*).

```
import java.awt.*;
```

- All classes in the `java.lang` package are automatically imported into all programs.
  - This includes `String` and `System` (among others)

Think-Pair-Share

- What is the difference between a class and an object?
- What does it mean to instantiate an object? How do you do this?
- What is a reference variable?
- What does a variable reference if the object it is supposed to reference is not instantiated?
- How do you tell an object to perform an action?

Intro to Java Graphics

- The `Graphics` class from the `java.awt` package is a useful class for drawing shapes on a canvas.
  - See the Intro to Graphics notes for details on how to use the `Graphics` class.
  - `http://cs.boisestate.edu/~cs121/notes/graphics-handout.pdf`

The String Class

- In Java, strings are immutable: Once we create a String object, we cannot change its value or length.
- The String class provides several useful methods for manipulating String objects. Many of these return a new String object since strings are immutable. For example:

```
String babyWord = "googoo";
String str = babyWord.toUpperCase();
```

- See javadocs for String for list of available methods: http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Selected Methods in String class

`int length()` `char charAt (int index)`

`String toLowerCase()`
`String toUpperCase()`
`String trim()`

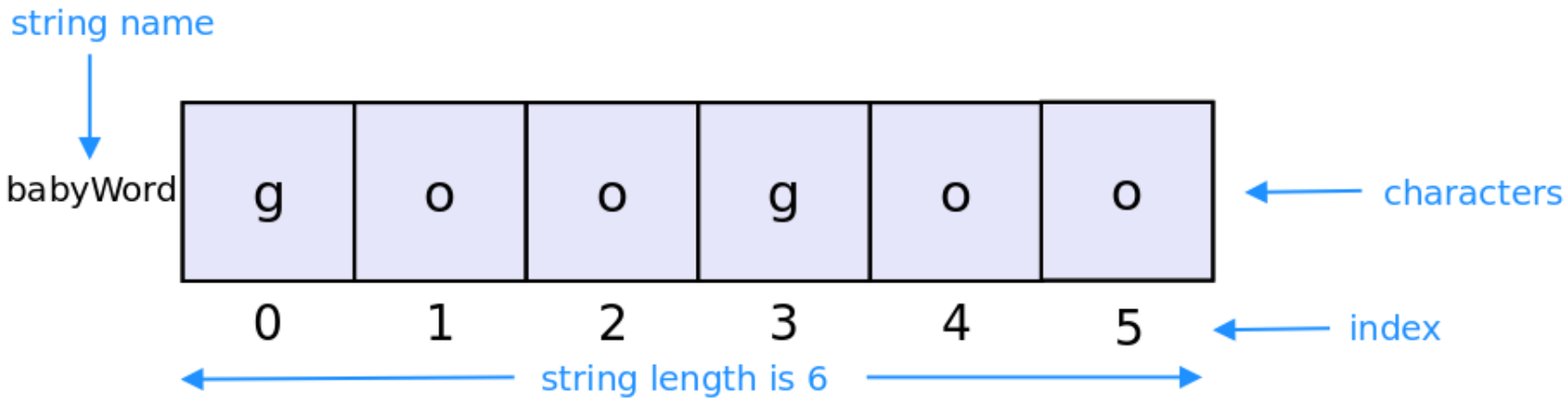
`boolean equals(String str)`
`boolean equalsIgnoreCase(String str)`
`int compareTo(String str)`

`String concat(String str)`
`String replace(char oldChar, char newChar)`
`String substring(int offset, int endIndex)`
returns a string that equals the substring from index `offset` to `endIndex - 1`

`int indexOf(char ch)`
`int indexOf(String str)`
returns the index of the first occurrence of character `ch` or string `str`

String Representation

- The `String` class represents a string internally as a series of characters. These characters have an index that we can use to refer to a specific character.



- We can use the `charAt(int index)` method to get the character at the `index` position.

```
char ch = babyWord.charAt(0);
char ch = babyWord.charAt(4);
```

String Examples

- Example: StringPlay.java.
- What output is produced by the following code?

```
String babyWords = "googoo gaagaa";
System.out.println(babyWords.length());
System.out.println(babyWords.toUpperCase());
System.out.println(babyWords.substring(7, 10));
System.out.println(babyWords.replace('g', 'm'));
System.out.println(babyWords.length());
```

The Random Class

- The `Random` class provides methods that generate pseudorandom numbers. The class is part of the `java.util` package.
- True random numbers are usually generated from nature or physical processes.
- Give some examples of physical processes that generate random numbers:
  - Flipping a coin
  - Rolling dice
  - Shuffling playing cards
  - Brownian motion of molecules in a liquid
- Pseudorandom numbers are generated using algorithms that start with a seed value. The values generated pass statistical tests. There are two main advantages of pseudorandom numbers:
  - Unlimited supply
  - Reproducibility
- Random numbers are used in simulations, security, testing software, design, games and many other areas.

Selected Methods in the `Random` class

`Random Random()` Constructor: creates a new pseudorandom generator
`Random Random(long seed)` Constructor: with a seed value to be able to reproduce random sequence
`int nextInt(int bound)` returns a random number over the range 0 to `bound-1`
`int nextInt()` returns a random number over all possible values of `int`
`double nextDouble()` returns a `double` random number between 0.0 (inclusive) and 1.0 (exclusive)

Using the Random Class

- Import the class, construct an instance and then use the appropriate methods.

```
import java.util.Random;
Random generator = new Random();
System.out.println(generator.nextInt(10));
System.out.println(generator.nextInt(10));
```

- Use the constructor with a seed argument to create a pseudorandom number sequence that is the same each time:

```
import java.util.Random;
long seed = 12345; //arbitrary number!
Random generator = new Random(seed);
System.out.println(generator.nextInt(10));
System.out.println(generator.nextInt(10));
```

- Example: RandomNumbers.java

In-class Exercises

- Given an `Random` object named `generator`, what range of values are produced by the following expressions?

```
generator.nextInt(25)
generator.nextInt(10) + 1
generator.nextInt(50) + 100
generator.nextInt(10) - 5
generator.nextInt(21) - 10
```

- Write an expression using `generator` that produces the following range of random values:

  0 to 12

  1 to 100

  15 to 20

  -10 to 0

- Create a random color using the `Color` class and the `Random` class.

The Math Class

- The `Math` contains methods for basic mathematical operations like exponentiation, square root, logarithm and trigonometric functions.
- Part of the `java.lang` package so no need to import.
- The methods in the `Math` class are static methods (also known as class methods).
- Static methods can be invoked using the class name — no `Math` object needs to be instantiated. For example:

```
double value = Math.sin(Math.PI) + Math.cos(Math.PI);
```

- Example: Quadratic.java, TrigDemo.java.

Selected Methods in the Math class

```
static int abs(int num)
static double sqrt(double num)
static double ceil(double num)
static double floor(double num)
```

```
static double log(double num)
static double log10(double num)
static double pow(double num, double power)
```

```
static double min(double num1, double num2)
static double max(double num1, double num2)
static int min(int num1, int num2)
static int max(int num1, int num2)
```

```
static double sin(double angle)
static double cos(double angle)
static double tan(double angle)
static double toRadians(double angleInDegrees)
static double toDegrees(double angleInRadians)
```

Formatting Output (1)

- The `java.text` package provides classes to format values for output.
  - The `NumberFormat` allows us to format values as currency or percentage.
  - The `DecimalFormat` allows us to format values based on a pattern.
- Two code code snippets that shows the usage of `NumberFormat` class (Note that the import statement will be at the top of the Java source file):
  - ```
    import java.text.NumberFormat;
    NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
    double amount = 1150.45;
    System.out.println("Amount:  " + fmt1.format(amount));
    ```
  - ```
    import java.text.NumberFormat;
    NumberFormat fmt2 = NumberFormat.getPercentInstance();
    double passRate = .8845;
    System.out.println("Amount:  " + fmt2.format(passRate));
    ```
- Example: Purchase.java

Formatting Output (2)

- The `DecimalFormat` allows us to format values based on a pattern.
  - For example, we can specify the number should be rounded to three digits after the decimal point.
  - Uses Half Even Rounding to truncate digits: round towards the "nearest whole neighbor" unless both whole neighbors are equidistant, in which case, round towards the even neighbor. See here for details: http://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html
- A code snippet that shows the usage:

```
import java.text.DecimalFormat;
DecimalFormat fmt = new DecimalFormat("0.###");
double amount = 110.3424;
System.out.println("Amount:  " + fmt.format(amount));
//shows 110.342
```

- We can change the rounding mode with the `setRoundingMode` method:

```
fmt.setRoundingMode(RoundingMode.CEILING);
System.out.println("Amount:  " + fmt1.format(amount));
//shows 110.343
```

Formatting Output (3)

- Commonly used symbols in the pattern:

0	digit (`int`, `short`, `byte`)
#	digit, zero shows as absent
.	decimal separator
,	grouping separator (for large numbers)
E	show in scientific notation

- Example: CircleStatsDecimalFormat.java
- **In-class exercise** What do the following patterns accomplish?

```
"##.###"
"00.###"
"###,###"
"000,000"
```

- We can set minimum and maximum limits on integer and fractional digits. For more information, see the javadocs for the `DecimalFormat` class.

Formatting Output (4)

- The class `Formatter` from the `java.util` package provides an alternative way of formatting output that is inspired by the `printf` method in C language.

```
import java.util.Formatter;
Formatter fmt = new Formatter(System.out);
double area = 1150.45;
fmt.format("The area is %f\n",area);
```

- Here the `%f` is a conversion template that says to format the variable `area` as a floating point number and insert in the output. Various conversions are available for printing a wide variety of types.
- Convenience methods exist in the `System.out` object to use `Formatter` class methods.

```
System.out.printf("The area is %f\n",area);
```

- We can also format a `String` object, which often comes in handy.

```
String output = String.format("The area is %f\",area);
```

- In each case, the underlying method used is the same.

Selected `printf` Style Formatting Conversions

- Commonly used *conversions*:

`%d`	decimal (`int`, `short`, `byte`)
`%ld`	`long`
`%f`	floating point (`float`, `double`)
`%e`	floating point in scientific notation
`%s`	`String`
`%b`	`boolean`

- Some examples of variations on the default formatting:

`%10d`	use a field 10 wide (right-aligned for numeric types)
`%8.2f`	use a field 8 wide, with two digits after the decimal point
`%-10s`	left justified string in 10 spaces (default is right justified)

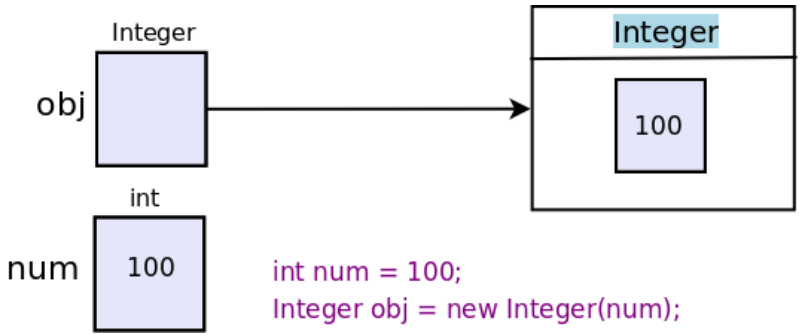
- Note that if the output doesn't fit in the number of spaces specified, the space will expand to fit the output.
- Examples: CircleStatsFormatter.java, CircleStatsPrintfTable.java, PrintfExample.java

Wrapper Classes (1)

- The `java.lang` package contains wrapper classes corresponding to each primitive type.

`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
`void`	`Void`

- See below for the relationship between the wrapper object and the primitive type:



- An object of a wrapper class can be used any place where we need to store a primitive value as an object.

Wrapper Classes (2)

- The wrapper classes contain useful static methods as well as constants related to the base primitive type.
- For example, the minimum `int` value is `Integer.MIN_VALUE` and the maximum `int` value is `Integer.MAX_VALUE`.
- Example: PrimitiveTypes.java
- For example, the `parseInt` method converts an integer stored as a `String` into an `int` value. Here is a typical usage to convert input from a user to an integer.

```
Scanner scan = new Scanner(System.in);
String input = scan.nextLine();
int num = Integer.parseInt(input);
```

Wrapper Classes (3)

- Selected methods from the `Integer` class.

`Integer(int value)` Constructor: builds a new `Integer` object that stores the specified value.
`static parseInt(String s)` Returns an `int` value corresponding to the value stored in the string `s`.

`static toBinaryString(int i)`
`static toOctalString(int i)`
`static toHexString(int i)`
Returns the string representation of integer `i` in the corresponding base.

- Similar methods and many more are available for all the wrapper classes. Explore the javadocs for the wrapper classes.

Autoboxing

- Autoboxing is the automatic conversion of a primitive value to a corresponding wrapper object.

```
Integer obj;
int num = 100;
obj = num;
```

- The assignment creates the corresponding wrapper `Integer` object. So it is equivalent to the following statement.

```
obj = new Integer(num);
```

- The reverse conversion (unboxing) also happens automatically as needed.

Summary

- Understand the difference between primitive type variables and reference variables.
- Creating and using objects.
- Using `String`, `Math`, `Random`, `Scanner` classes.
- Formatting output using `NumberFormat`, `DecimalFormat` and `Formatter` classes.
- Wrapper classes and autoboxing: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Character`, `Boolean`

Exercises

- Read Chapter 3.
- **Recommended Homework**:
  - Exercises: EX 3.2, 3.3, 3.4, 3.6, 3.7, 3.11, 3.12.
  - Projects: PP 3.2, 3.3, 3.5.