CSC 243 - Java Programming

Java Data Types and Control Constructs

Java Types

- In general, a type is collection of possible values
- Main categories of Java types:
    - Primitive/built-in
    - Object/Reference

Java Built-in Types

- **byte** 8-bit signed
- **short** 16-bit signed
- **int** 32-bit signed
- **long** 64-bit signed
- **float** 32-bit
- **double** 64-bit
- **boolean** true or false
- **char** Unicode character
- **String** a built-in *class* representing a sequence of characters

Java Object/Reference Types

- Object types are accessed via a reference

  ```
  Object a = new Object();
  Object b = a;
  ```

- The assignment operator copies the reference

Java Object Construction

- The *constructor* has the same name as the class
- Any method, including the constructor, can be overloaded
- The `new` operator creates a new instantiation of an object using an object constructor and returns a reference to that object

Wrapper Classes

- Wrapper classes provide a mechanism to convert between primitive and object types
- Wrapper class list:

Primitive Type	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Conversion Examples

Convert `int` into `Integer`

```
int x = 1;
Integer i = Integer.valueOf(x);
Integer j = x;
```

Convert `Integer` into `int`

```
Integer x = new Integer(1);
int i = x.intValue();
int j = x;
```

Wrapper classes and String

Wrapper classes can also be used to convert String types to Primitive types:

```
String s = "3";
int i = Integer.parseInt(s);
```

Basic Exception Handling

- `Integer.parseInt` signature:

```
public static int parseInt(String s)
    throws NumberFormatException
```

- Handle the exception:

```
try {
  String s = "3";
  int i = Integer.parseInt(s);
}
catch (NumberFormatException e) {
  // code to handle the exception
}
```

Java Control Flow Constructs

- The basis for control flow is the `boolean` type
- `for`, `while`, and `do while` loops
- `if` and `else` selection
- `switch` statements
- `break` exits the inner-most loop or switch
- `continue` jumps to the next iteration of the loop

Boolean Operators

- Logical operators
  - and: `&&`
  - or: ||
  - not: `!`
- Comparison operators
  - equal: ==
  - not equal: `!=`
  - less than: <
  - less than or equal: <=
  - greater than: >
  - greater than or equal: >=

Object Comparison

- == compares the references – returns true if both operands refer to the same object
- `object.equals()` compares objects using an class defined method

```
String s1 = new String("S");
String s2 = s1;
s1 == s2; // true
s1.equals(s2); // true
s1 = new String("S");
s1 == s2; // false
s1.equals(s2); // true
```

Java Static Methods

- A method is a function associated with an object
- A *static* method does not require an object instance

```
int x = java.lang.Integer.parseInt("3");
System.out.println("x: " + x);
```

- An *instance* method requires an object reference

```
Integer x = new Integer(3);
System.out.println("x: " + x.intValue());
```

Java Static and Non-static Data Fields

- Only one copy of a static data field exists
- Both static and instance methods can use a static data field
- For a non-static data field, there is one copy for each instantiated object
- Only a non-static method can use a non-static data field

Java Access Modifiers

- **public** methods and data can be used by any code that imports the class
- **protected** methods and data can be used only by the defining class and derived classes
- **private** methods and data can be used by the defining class imports the class
- If there is no explicit access restriction, then the methods and data can be used by any class in the same **package**

Java Arrays

- Arrays are constructed using the `new` operator
- Arrays are initialized based on type:
  - numeric types (int, float, etc.) are initialized to zero
  - booleans are initializede to false
  - chars are initialized to '\0000'
  - objects are initialized to null
- Examples:

```
float[] numbers = new float[10];
int[] counts = {1, 2, 3};
Object[] objects = new Object[20];
```

Basic Java Array Usage

- Access an element with the `[]` operator; if the index is outside of the array, a `ArrayIndexOutOfBoundsException` is thrown
- The `length` property contains the size of the array
- Looping over arrays:

```
int[] numbers = {1, 2, 3};

// for loop
for (int i = 0; i < numbers.length; i++) {
  System.out.println(i);
}

// foreach loop
for (int element: numbers) {
  System.out.println(element);
}
```