Dostęp do wybranego elementu witryny najłatwiej uzyskać za pomocą metody getElementById obiektu document. Oczywiście warunkiem jest to, aby podczas definiowania takiego elementu nadać mu unikatowy atrybut id. Przykładowo dostęp do poszczególnych składowych formularza wcale nie musi się odbywać poprzez konstrukcję typu:

document.forms.nazwa_formularza.nazwa_elementu

Co więcej, w niektórych przypadkach nie ma nawet potrzeby definiowania samego obiektu formularza. Warto zauważyć, że w przykładzie zliczającym wystąpienia danego ciągu w tekście był definiowany formularz, dzięki czemu możliwy był dostęp do elementów składowych. Niemniej ten formularz był wykorzystywany tylko lokalnie (nie był używany do zbierania danych i wysyłania ich do serwera). Jeśli więc w celu dostępu do pól tekstowych zostanie zastosowana metoda getElementById, znacznik <form> stanie się zupełnie zbędny. Kod przedstawiony na listingu 5.16 pokazuje, jakie zmiany należy wprowadzić (a dotyczą one zarówno części HTML, jak i JavaScript), aby zastosować opisany sposób dostępu.

Listing 5.16. Bezpośredni dostęp do elementów witryny

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;

 &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>

<html lang=&quot;pl&quot;>

 <head>

  <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>

  <meta http-equiv=&quot;Content-Script-Type&quot; content=&quot;text/javascript&quot;>

  <title>Moja strona WWW</title>

  <script type=&quot;text/javascript&quot;>

  function szukaj()

  {

    var tekst = document.getElementById('okno_tekstowe').value;

    var ciag = document.getElementById('ciag_znakow').value;

    if((ciag == &quot;&quot;) || (tekst == &quot;&quot;)) return;

    var indeks = 0;

    var indeks_wystapienia = 0;

    var liczba_wystapien = 0;



    while (indeks_wystapienia != -1){
```

```
      indeks_wystapienia = tekst.indexOf(ciag, indeks);
      if (indeks_wystapienia != -1){
        indeks = indeks_wystapienia + ciag.length;
        liczba_wystapien++;
      }
    }
    document.getElementById('ile').value = liczba_wystapien;
  }
  </script>
 </head>
 <body>
  <table border=&quot;0&quot;>
  <tr>
   <td>Wpisz w tym oknie tekst:
   </td>
   <td>Wpisz szukany ciąg znaków:
   </td>
  </tr>
  <tr>
   <td rowspan=&quot;2&quot;>
    <textarea id=&quot;okno_tekstowe&quot;
              cols=&quot;35&quot;
              rows=&quot;10&quot;></textarea>
   </td>
   <td valign=&quot;top&quot;  align=&quot;right&quot;>
    <input type=&quot;text&quot;
           id=&quot;ciag_znakow&quot;>
    <br /><br />
```

```
      <input type   =&quot;button&quot;

            value   =&quot;Szukaj&quot;

            onClick=&quot;szukaj();&quot;>

    </td>

   </tr>

   <tr>

    <td align=&quot;right&quot;>

      Podany ciąg znaków wystąpił w  tekście:<br />

      <input type=&quot;text&quot;  id=&quot;ile&quot;><br />

      razy.

    </td>

   </tr>

   </table>

  </body>

</html>
```

Tym razem nie ma znacznika <form>, natomiast wszystkie pola tekstowe otrzymały unikatowe (niepowtarzające się) identyfikatory:

-

okno_tekstowe — pole służące do wprowadzania tekstu;

-

ciag_znakow — pole służące do wprowadzenia ciągu, którego wystąpienia będą zliczane;

-

ile — pole służące do prezentacji wyników.

Zmodyfikowana została również funkcja szukaj. Dostęp do wprowadzonych przez użytkownika danych jest teraz realizowany przez wywołania metody getElementById. Każde takie wywołanie zwraca obiekt o identyfikatorze przekazanym w postaci argumentu, w tym przypadku konkretne pole tekstowe, posiadające właściwość value zawierającą przechowywany przez nie tekst. Samo przeszukiwanie tekstu jest realizowane tak jak w poprzednim przykładzie.

W rozdziale opisującym struktury języka JavaScript prezentowany był skrypt obliczający rozwiązania równania kwadratowego. Wymagał on jednak, aby parametry tegoż równania były podane bezpośrednio w kodzie. Obecnie, kiedy zostały już zaprezentowane elementy umożliwiające wprowadzanie danych przez użytkownika, można wykonać przykład realizujący takie zadanie, ale pozwalający na wygodne rozwiązywanie dowolnych równań.

Na stronie zostaną umieszczone cztery pola tekstowe, trzy służące do wprowadzania danych i jedno do przedstawiania rozwiązań, tak jak jest to widoczne na rysunku 5.5.

Rysunek 5.5. Skrypt rozwiązujący równania kwadratowe

Każde pole otrzyma identyfikator, który pozwoli na dostęp do niego z poziomu skryptu. Te identyfikatory to:

- paramA — dla pola przechowującego parametr A,
- paramB — dla pola przechowującego parametr B,
- paramC — dla pola przechowującego parametr C,
- wynik — dla pola przechowującego wynik.

Obliczenie wyniku będzie się odbywało po kliknięciu widocznego na rysunku przycisku Oblicz, cały skrypt będzie zaś miał postać zaprezentowaną na listingu 5.17.

Listing 5.17. Obliczanie rozwiązań równania kwadratowego o zadanych parametrach

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;
  &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>
<html lang=&quot;pl&quot;>
 <head>
   <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>
   <meta http-equiv=&quot;Content-Script-Type&quot; content=&quot;text/javascript&quot;>
   <title>Moja strona WWW</title>
   <script type=&quot;text/javascript&quot;>
   function oblicz()
   {
     //deklaracje zmiennych odpowiadających parametrom równania
     var A = parseInt(document.getElementById(&quot;paramA&quot;).value);
     var B = parseInt(document.getElementById(&quot;paramB&quot;).value);
```

```
var C =  parseInt(document.getElementById(&quot;paramC&quot;).value);

var wynikTekst = &quot;&quot;;


//sprawdzenie,  czy jest to równanie kwadratowe

if (A == 0){

  //A  jest równe zero, równanie nie jest kwadratowe

  wynikTekst = &quot;To nie jest równanie kwadratowe: A  = 0!&quot;;

}

else{

  //A  jest różne od zera, równanie jest kwadratowe


  //obliczenie  delty

  var delta = B * B - 4 * A * C;


  //jeśli  delta mniejsza od zera

  if (delta < 0){

    wynikTekst = &quot;Delta < 0. To równanie nie ma  rozwiązania &quot;;

    wynikTekst += (&quot;w zbiorze liczb  rzeczywistych!&quot;);

  }

  //jeśli  delta równa zero lub większa od zera

  else{

    //jeśli delta jest równa zero

    if (delta == 0){

      //obliczenie wyniku

      var wynik = - B / 2 * A;

      wynikTekst = &quot;Rozwiązanie: x = &quot; +  wynik;

    }

    //jeśli delta jest większa od zera
```

```
        else{
          //obliczenie wyniku
          wynik = (- B + Math.sqrt(delta)) / 2 * A;
          wynikTekst = &quot;x1 = &quot; + wynik;
          wynik = (- B - Math.sqrt(delta)) / 2 * A;
          wynikTekst += &quot;, x2 = &quot; + wynik;
        }
      }
    }
    document.getElementById('wynik').value = wynikTekst;
  }
  </script>
</head>
<body>
  <div>
  parametr A:
  <input type=&quot;text&quot;  id=&quot;paramA&quot; size=&quot;3&quot;>
  parametr B:
  <input type=&quot;text&quot;  id=&quot;paramB&quot; size=&quot;3&quot;>
  parametr C:
  <input type=&quot;text&quot;  id=&quot;paramC&quot; size=&quot;3&quot;>
  <input type   =&quot;button&quot;
        value  =&quot;Oblicz&quot;
        onClick=&quot;oblicz();&quot;
  >
  <br /><br />
  Wynik:
  <input type=&quot;text&quot;  id=&quot;wynik&quot; size=&quot;70&quot;>
```

```
    </div>

  </body>

</html>
```

Funkcja oblicz, która jest procedurą obsługi zdarzenia onclick przycisku Oblicz, wykonuje bardzo podobne zadanie jak prezentowany już wcześniej skrypt, wszak zasady rozwiązywania równań kwadratowych pozostały niezmienione. Niemniej występuje w niej kilka modyfikacji w stosunku do wspomnianego przykładu. Przede wszystkim parametry równania odzwierciedlane przez zmienne A, B i C są odczytywane z pól tekstowych, do których dostęp jest zapewniany przez metodę getElementById. Każdy ciąg znaków odczytywany z pola tekstowego jest zamieniany na wartość liczbową za pomocą funkcji parseInt.

Wynik obliczeń jest z kolei zapisywany w zmiennej pomocniczej wynikTekst i dopiero na samym końcu funkcji oblicz wartość tej zmiennej jest zapisywana w polu tekstowym o identyfikatorze wynik, dzięki czemu dane pojawiają się na ekranie.

Dostęp do elementów witryny umożliwia też np. napisanie skryptu, który wyświetli aktualną godzinę na zwykłym przycisku generowanym za pomocą znacznika input, tak jak jest to widoczne na rysunku 5.6.

Rysunek 5.6. Zegar na przycisku

Uzyskanie aktualnego czasu jest możliwe dzięki obiektowi Date, którego opis został zawarty we wcześniejszych rozdziałach. Wywołania metod getHours, getMinutes i getSeconds pozwalają na skonstruowanie ciągu zawierającego aktualny czas, tak jak zostało to wykonane w funkcji showTime widocznej na listingu 5.18.

Listing 5.18. Aktualny czas wyświetlany na przycisku

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;

  &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>

<html lang=&quot;pl&quot;>

 <head>

   <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>

   <meta http-equiv=&quot;Content-Script-Type&quot; content=&quot;text/javascript&quot;>

   <title>Moja strona WWW</title>

   <script type=&quot;text/javascript&quot;>

   function showTime()

   {

     var data = new Date();

     var godziny = data.getHours();
```

```
    var minuty = data.getMinutes();
    var sekundy = data.getSeconds();
    var czas = godziny;
    czas += ((minuty < 10) ?  &quot;:0&quot; : &quot;:&quot;) + minuty;
    czas += ((sekundy < 10) ? &quot;:0&quot; :  &quot;:&quot;) + sekundy;
    document.getElementById(&quot;btnCzas&quot;).value =  czas;
  }
  setInterval(&quot;showTime()&quot;,  1000);
  </script>
 </head>
 <body>
  <div>
  <input type   =&quot;button&quot;
        value  =&quot;--:--:--&quot;
        id=&quot;btnCzas&quot;
  >
  </div>
 </body>
</html>
```

Czas ten jest wyświetlany na elemencie witryny o identyfikatorze btn1, którym jest zwykły przycisk. Oczywiście dostęp do tego elementu jest uzyskiwany dzięki wywołaniu metody getElementById obiektu document. Cykliczne wywoływanie funkcji showTime, uaktualniające pokazywany czas, zapewnia z kolei wywołanie metody setInterval obiektu window.

Równie ciekawym efektem jest cykliczna zmiana koloru przycisku, aby jednak był on atrakcyjny, zmiany te nie powinny być przypadkowe — przejścia między barwami powinny być możliwie płynne. Aby to osiągnąć, najwygodniej wygenerować programowo odpowiednią tabelę kolorów, z której następnie będą pobierane dane. Jak to zrobić, zostało zobrazowane na listingu 5.19.

Listing 5.19. Przycisk zmieniający kolor

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;
 &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>
<html lang=&quot;pl&quot;>
```

```
<head>
  <meta  http-equiv=&quot;Content-Type&quot; content=&quot;text/html;  charset=utf-8&quot;>
  <meta  http-equiv=&quot;Content-Script-Type&quot;  content=&quot;text/javascript&quot;>
  <title>Moja strona  WWW</title>
  <script  type=&quot;text/javascript&quot;>
  var speed = 1;
  var colorTable = new Array();
  var index = 0;
  function generujKolory()
  {
    for (i = 0; i < 256; i++){
      var color = &quot;#&quot;;
      color += &quot;00&quot;;
      color += (i < 16)?&quot;0&quot; +  i.toString(16):i.toString(16);
      color += ((255-i) <  16)?&quot;0&quot; + (255-i).toString(16):(255-i).toString(16);
      colorTable[i] = color;
      color = &quot;#&quot;;
      color += &quot;00&quot;;
      color += ((255-i) <  16)?&quot;0&quot; + (255-i).toString(16):(255-i).toString(16);
      color += (i < 16)?&quot;0&quot; +  i.toString(16):i.toString(16);
      colorTable[i + 256] = color;
   }
  }
  function zmienKolor()
  {
     document.getElementById(&quot;btn1&quot;).style.background =
      colorTable[index++];
    if(index >= colorTable.length) index  = 0;
```

```
   }

   function start()

   {

     generujKolory();

     setInterval(&amp;quot;zmienKolor()&amp;quot;,  speed);

   }

   </script>

  </head>

  <body onload=&amp;quot;start();&amp;quot;>

    <div>

    <input type=&amp;quot;button&amp;quot;

          value=&amp;quot;Przycisk&amp;quot;

          id=&amp;quot;btn1&amp;quot;

    >

    </div>

  </body>

</html>
```

Za cykliczną zmianę stylu przycisku odpowiada funkcja zmienKolor. Prędkość zmian jest natomiast regulowana zdefiniowaną na początku skryptu zmienną speed, która określa, co ile milisekund ma następować wywołanie zmienKolor. Generowaniem tablicy kolorów (colorTable) zajmuje się funkcja generujKolor, natomiast cały proces jest zapoczątkowywany w funkcji start, która jest procedurą obsługi zdarzenia onload sekcji body. Funkcja start najpierw wywołuje generujKolory, a następnie rozpoczyna działanie timera przez wykonanie instrukcji:

```
setInterval(&amp;quot;zmienKolor()&amp;quot;, speed);
```

Tabela przejść między barwami, tworzona w funkcji generujKolory, może być wykonana na różne sposoby. W tym konkretnym przypadku cykl rozpoczyna się od koloru niebieskiego (#0000FF), który płynnie przechodzi w zielony (#00FF00), a ten z powrotem w niebieski (#0000FF). W każdym przebiegu pętli for są więc modyfikowane dwie składowe RGB.

W całej tabeli składowa G zmienia się w zakresie: 00 – FF – 00, B w zakresie: FF – 00 – FF, a R pozostaje bez zmian (cały czas jest równa 0). Dodatkową komplikacją jest konieczność dokonania konwersji zmiennej i na ciąg znaków reprezentujący jej wartość w postaci szesnastkowej. Jest to wykonywane za pomocą funkcji toString.

Parametrem tej funkcji jest podstawa systemu liczbowego, w jakim ma być zwrócony wynik. Przykładowo: jeśli jest zadeklarowana zmienna i zawierająca wartość 255, to wykonanie instrukcji:

```
var napis = i.toString(16);
```

spowoduje przypisanie do zmiennej napis ciągu znaków FF.

To jednak nie rozwiązuje do końca problemu. Otóż kolor musi być zdefiniowany w postaci #RRGGBB, czyli kolor niebieski jako #0000FF (R = 0, G = 0, B = 255). Tymczasem wykonanie instrukcji:

var napis = "#" + (0).toString(16) + (0).toString(16) + (255).toString(16);

dałoby w rezultacie ciąg znaków #00FF. Jest to zrozumiałe, jako że liczby od 0 do 15 dziesiętnie mają w postaci szesnastkowej postać jednocyfrową od 0 do F. Należy zatem uwzględnić ten fakt podczas konwersji i jeśli to konieczne, dodawać brakujące zera. Najwygodniej jest skorzystać z wyrażenia warunkowego w postaci:

(wyrażenie warunkowe) ? wartość1 : wartość2

Jeśli wyrażenie jest prawdziwe, całość przyjmuje wartość1, w przeciwnym razie wartość2. Dlatego do dokonania odpowiedniej konwersji jest używana konstrukcja:

(i < 16) ? "0" + i.toString(16) : i.toString(16);

Funkcja zmienKolor działa w bardzo prosty sposób. Uzyskuje dostęp do przycisku za pomocą metody getElementById oraz odwołuje się do właściwości style.background odpowiadającej kolorowi tła przycisku. Tej właściwości przypisuje też wartość pobraną z tablicy colorTable zawierającej definicje wygenerowanych wcześniej kodów kolorów. Aktualny kod wskazywany jest przez zmienną index, która jest zwiększana przy każdym wywołaniu funkcji zmienKolor (index++). Gdy index osiągnie wartość równą rozmiarowi tablicy colorTable, jest zerowany, a tym samym rozpoczyna się kolejny cykl.